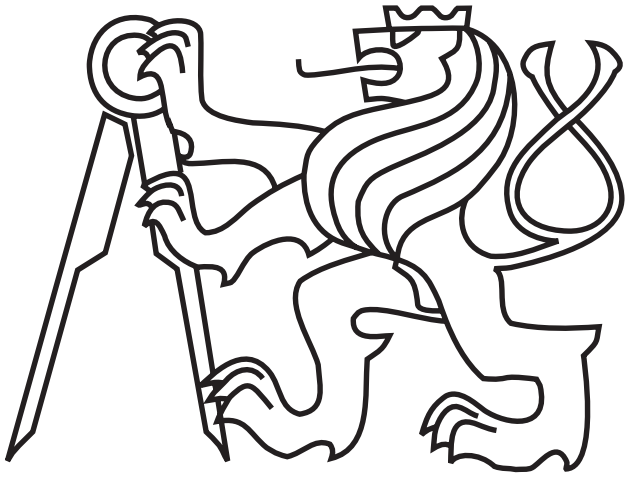
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ

DIPLOMOVÁ PRÁCE



Bc. Michal Marek

Sledování pohyblivých objektů v obraze z helikoptéry

Katedra kybernetiky
Vedoucí diplomové práce: **Ing. Jan Chudoba**

Květen 2016

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne ....

podpis autora práce

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Michal M a r e k

Studijní program: Kybernetika a robotika

Obor: Robotika

Název tématu: Sledování pohyblivých objektů v obraze z helikoptéry

Pokyny pro vypracování:

Prostudujte problematiku počítačového zpracování obrazu se zaměřením na nalezení korespondence mezi obrázky pořízenými z různých míst a na detekci změn v následujících obrazech z kamery.
Na základě vyhodnocení dostupných metod a výsledků projektů na jiných pracovištích navrhněte metodu detekce změn v obraze vhodnou pro detekci objektů v obraze snímaném helikoptérou, která stojí ve vzduchu, nebo se pomalu pohybuje. Přijměte potřebná zjednodušující opatření s ohledem na řešitelnost projektu, jako např. konstantní výška letu, nebo omezené otáčení helikoptéry.
Navrženou metodu implementujte a proveďte vyhodnocení její funkce na datech pořízených reálnou helikoptérou.

Seznam odborné literatury:

[1] Qian Yu and Gérard Medioni: "A GPU-based implementation of Motion Detection from a Moving Platform", in IEEE workshop on Computer Vision on GPU, in conjunction with CVPR´08.
[2] Rodríguez-Canosa, G.R.; Thomas, S.; del Cerro, J.; Barrientos, A.; MacDonald, B.: A Real-Time Method to Detect and Track Moving Objects (DATMO) from Unmanned Aerial Vehicles (UAVs) Using a Single Camera. Remote Sens. 2012, 4, 1090-1111.

Vedoucí diplomové práce: Ing. Jan Chudoba

Platnost zadání: do konce letního semestru 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 19. 2. 2016

Abstrakt

Diplomová práce se zabývá problémem snímání okolního prostředí dynamicky se pohybující kamerou a následnou detekcí pohybujících se objektu. Vývoj probíhal v programovacím jazyku C++ za použití volně dostupné knihovny OpenCV. Reálný hardware, který obsahuje drona, byl nahrazen notebookem acer Travelmate, který je nízkým výkonem srovnatelný s výpočetním výkonem drona. Snímání okolního prostředí bylo zprostředkováno kamerou. Z dat získaných z tohoto senzoru, byl detekován pohyb vůči okolnímu prostředí pomocí Feature bodů. Tím byl získán pohyb vůči okolnímu prostředí. Následně byly detekovány změny prostředí pomocí dvou metod. První metoda využívá změn polohy Feature bodů a druhá metoda využívá nalezení rozdílu mezi dvěma sesazenými snímky. K tomuto účelu byla použita knihovna OpenCL. Cílem celé práce je detekovat pohybující se objekty ve snímaném prostředí.

Abstract

The thesis deals with the problem of scanning of the environment by dynamically moving camera and the detection of the moving object. Development proceeded in the programming language C++ with freely available library OpenCV. The real hardware, which includes drone was simulated by Acer TravelMate notebook, which has comparable low power consumption and computation power like drone. Scanning surrounding environment was realised by the camera. From the data obtained from this sensor, the motion in environment was detected by using Feature points. Thus the movement to the surrounding environment was obtained. Changes in the environment were detected using two methods. The first method utilizes changes in position Feature points and the second method finds the difference between two images deposed. For this purpose the OpenCL library was used.

Rád bych poděkoval mému vedoucímu diplomové práce panu Ing. Janu Chudobovi za cenné rady, vstřícnost, odborné vedení a pomoc při zpracování této práce. Dále bych chtěl poděkovat rodině za podporu během celého studia.

Obsah

Seznam tabulek

Seznam obrázků

Úvod

Jednou z motivací počítačového vidění a robotiky je soupeření s člověkem. Už od středověku se mnoho vědců pokoušelo postavit stroj, který bude vykazovat stejné vlastnosti jako člověk. Až doposud se to nepovedlo. Důvodem není technická nedokonalost materiálů či motorů, které jsou dneska poměrně vyspělé. Příčinou je nedostatečně pokročilá umělá inteligence, jejíž nedílnou součástí je vnímání a rozpoznávání okolního prostředí. Dnešní roboti disponují kamerami, které mají vyšší rozlišovací schopnost než lidské oko. člověk má ale výhodu, že disponuje mozkem, konkrétně šedou kůrou mozkovou, kde většina šedé kůry mozkové se orientuje na zpracování informací ze zraku. Lidský mozek je v řádu 1000-krát výkonnější než dnešní nejpokročilejší procesory či grafické karty. Proto bude trvat několik desetiletí, než bude robot na stejné úrovni jako člověk.
Tato práce se zabývá létajícím robotem (drona), který se pohybuje ve 3D prostoru a snímá okolní prostředí z ptačí perspektivy. Tyto létající stroje se rozlišují a nazývají podle počtu vrtulek, kterými disponují. Obecně platí, čím více vrtulek, tím je stroj stabilnější, ale řízení je složitější. V dnešní době autonomní létající stroje zaznamenávají obrovský rozmach, protože jsou snadno dostupné a levné. S tím přichází i řada bezpečnostních rizik, proto existují bezletové zóny (např. nad letištěm) kam tyto stroje nesmí. V některých zemích je dokonce nutné povolení k provozu těchto létajících strojů.
Téměř všechny kvalitnější drony obsahují stabilizátory polohy, výpočetní jednotku (některé mohou disponovat i GPU) a kameru, kterou snímat okolní prostředí. To je nezbytné pro tuto práci, protože úkolem je zpracovat video. Kvalita kamer umístěných na klasické droně nižší nebo střední třídy není příliš vysoká, protože je kladen důraz především na hmotnost a cenu kamery. Obraz produkovaný z těchto kamer je často zkreslený radiální chybou, proto je nutné tyto chyby opravit na základě správné kalibrace kamery.
Ke kalibraci kamery v této práci bude využita volně dostupná knihovna pro počítačové vidění OpenCV. Dalším problémem negativně ovlivňující kvalitu obrazu jsou vibrace, které vznikají nedokonalým vyvážením rotorů. Vibrace mají vyšší frekvenci než je frekvence kamery (počet snímků za sekundu) a výsledný obraz je pak rozmazaný, protože jednotlivé řádky v obrazu jsou snímány z jiné polohy a jsou vůči sobě navzájem posunuté. Jeden ze způsobů, jak eliminovat vibrace, je zavěšení kamery na odpružený závěs.

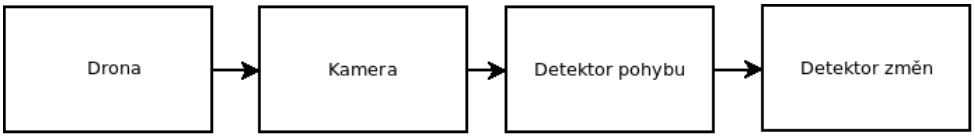
Úkolem drony je letět v daném prostředí a natáčet scénu pod sebou. Video projde analýzou a detekují se potencionální pohyblivé objekty. K tomuto účelu bude využita knihovna OpenCV, která sice primárně poskytuje funkce pro detekci pohyblivých předmětů, nicméně pouze pro statickou kameru, která se nepohybuje. Aby bylo možné detekovat pohybující objekty mobilní kamerou je nezbytná sesadit jednotlivé snímky a zjistit rychlost pohybu kamery. Tento úkol je realizován obrázkovými příznaky. Dalším úkolem je detekce pohyblivých objektů. Existuje celá řada metod, ale v této práci budou vyzkoušeny pouze dvě a to: analýza obrázkových příznaku a odečítací metoda. Příznaková analíza je rychlejší a hodí se především pro detekci velkých objektů, naopak odečítací metoda je výpočetně náročnější, ale je schopná detekovat i malé objekty. Pro dosažení real-timové rychlosti je nutné využít grafický procesor pomocí knihovny OpenCL.
Detekce pohybujících se objektů od mobilního agenta má obrovský potenciál a nabízí se celá řada využití, například ve vojenském průmyslu, bezpečnostních agentur až po zábavný průmysl. Existuje celá řada podobných prací zabývající se touto problematikou. Tento problém je tak rozsáhlý, že neexistuje univerzální metoda pro detekci pohyblivých objektů, proto každá práce popisuje odlišnou metodu. Cílem této práce je detekovat pohyblivé objekty přímo na droně v reálném čase.

Kapitola 1

Definice úlohy

1.1 Framework

Tato práce se skládá z těchto modulů: drona, kamera, detektor pohybu a detektor změn. Hierarchii můžete vidět v následujícím obrázku.



Obrázek 1.1: Schéma aplikace

Modul drona se stara pouze o řizení drony, které probíha buď autonomně, nebo tuto činnost zajišťuje pilot s dálkovým ovládáním. Důležité jsou regulátory polohy, které zajišťují stabilní polohu. Dále pak vyvážení jednotlivých rotorů, aby nedocházelo k vybracím, které opět negativně ovlivňují kvalitu záznamu.

Modul kamera se skládá jak z fyzické, tak ze softwarové části. Hardwarová část snímá okolní prostředí a pořizuje datový záznam, softwarová čast data upravuje a koreluje chybovost.

Modul detektor pohybu získáva upravená data z modulu kamera, pomocí těchto dat detekuje pohyb vůči okolnímu prostředí (detekce probíhá mezi jednotlivými snímky). Upravená data jsou následně odeslána do modulu detektor změn.

Modul detektor změn získává data jak z modulu kamera, tak z modulu detektor pohybu. Data jsou porovnávána pomocí několika algoritmů, pomocí níhž jsou detekovány obecné objekty, které jse v daném prostředí pohybují.

1.2 Počítačové vidění

Počítačové vidění je obor výpočetní techniky, který se zabývá získáváním dat a informací ze zachyceného obrazu či videa. Jde přitom o relativně mladý obor, který se začínal rozvíjet až počátkem 90. let, protože dříve byl tehdejší hardware příliš slabý nato, aby vykonával tak výpočetně náročné operace, které zahrnuje tento obor. Jako snad každé odvětví umělé inteligence, i zde dochází k inspiraci od biologického vidění, například u lidí je zrak považován za nejdůležitější smysl, který zaměstnává více než polovinu lidského mozku. Dosáhnout takové úrovně rozpoznávání objektů, jakou disponují zvířata či člověk je velice obtížné a ještě řadu let bude nedosažitelné.
Počítačové vidění je opakem tvorby počítačové grafiky, kde se z informací tvoří scéna. Opačný proces, tedy získávání informací z obrazu je složitější. Neexistují univerzální metody, jak řešit širokou škálu úloh, které zahrnuje tento obor. Existuje několik známých metod jak řešit daný problém, ovšem většinou to platí pro určité data a v jiných případech je nezbytné zvolit jinou metodu. Proto dochází k intenzivnímu výzkumu, protože v dnešní době tento obor nachází široké komerční uplatnění, ať už jde o průmysl, zábavu či medicínu [11].

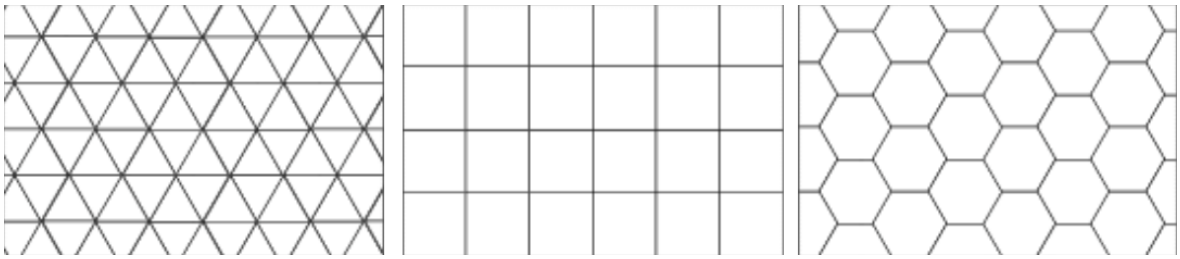
1.2.1 Digitalizace

Digitalizace je proces počítačového vidění, který převádí spojitou obrazovou funkci na diskrétní. Tím se převede definiční obor i obor hodnot. Jde o velice důležitý proces, protože například při natáčení videa by měl spojitý obraz příliž velikou velikost, proto je výhodnější zachytit určitý počet snímků podle definované frekvence. I lidské oko je schopné rozpoznávat omezený počet snímků za jednu sekundu. Dalším důležitým faktorem je rozlišení, což je počet pixelů daného snímku. Dnešním trendem je snaha rozlišení stále zvyšovat jak u videa tak u monitorů. Na nejmodernějších UHD monitoru je pixel menší než nejmenší bod, který je schopné lidské oko rozpoznat. Při digitalizaci tedy dojde ke ztrátě informace a téměř nikdy nelze úplně přesně zrekonstruovat původní obraz.

Vzorkování

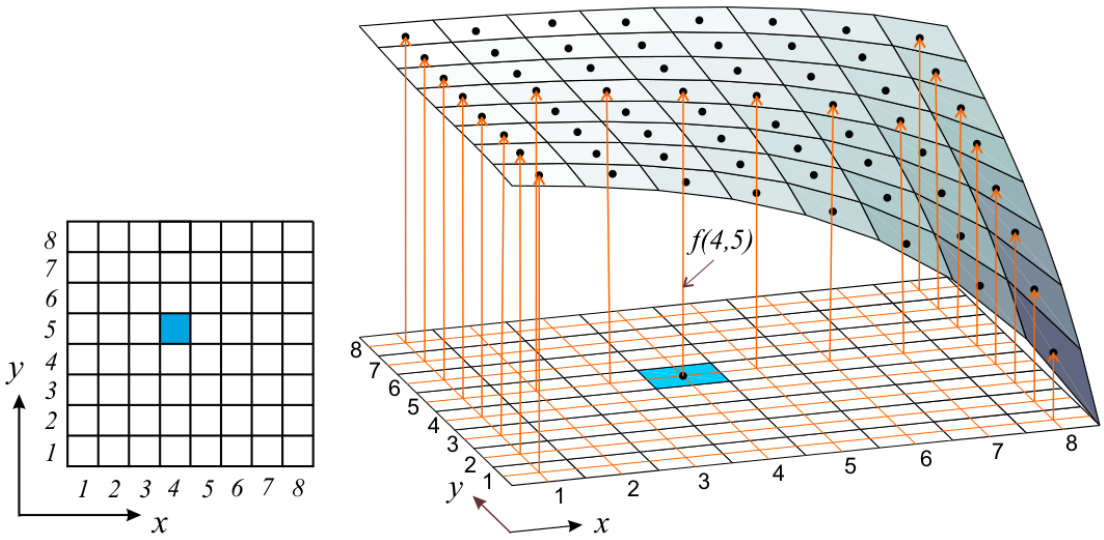
Vzorkování je proces, při kterém dochází k diskretizaci signálu či obrazu v časové oblasti. Základní elektronickou součástkou, která umožňuje vzorkování signálu je A/D převodník. Čím je vyšší vzorkovací frekvence, tím je možné pozorovat větší detaily. Důležitým pravidlem, který platí při vzorkování je Shanonův teorém, který říká: pro dosažení přesné rekonstrukce navzorkovaného signálu je nutné, aby byla zvolena minimálně vzorkovací frekvence dvakrát vyšší, než nejvyšší harmonická složka spojitého signálu. V praxi se používá dvojnásobek frekvence a malá rezerva (např. 5% vzorkovací frekvence). To se týká obecného spojitého signálu, nicméně vzorkování obrazu přináší mnohem víc problémů. Jedním z nich je způsob

uspořádání vzorkovací bodů (pixelů) do pravidelného rastru. Nejčastěji se využívá čtvercová mřížka, která je jednoduchá a technicky snadno proveditelná, ale v praxi se využívají i s trojúhelníkovou či hexagonální. Typy rastrů znázorňuje obrázek 1.2.

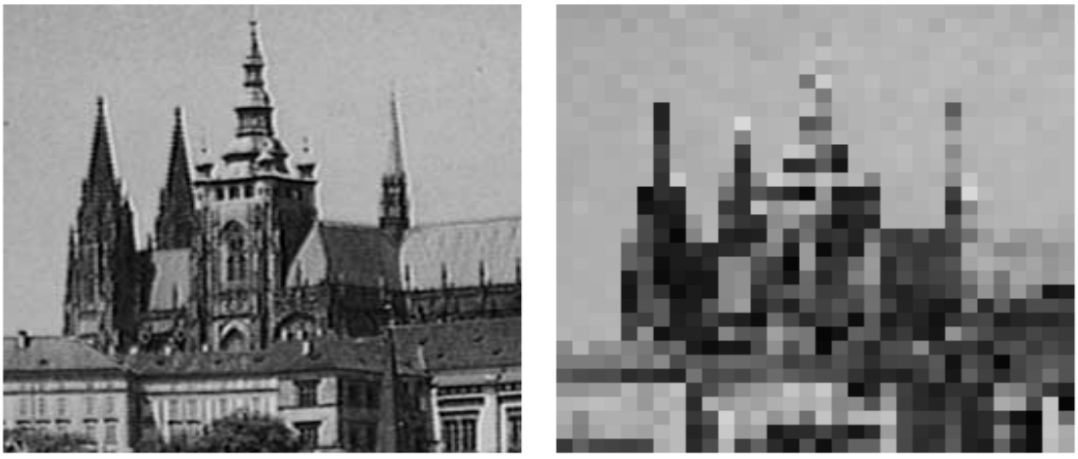


Obrázek 1.2: Typy rastrů [6].

Dalším problémem je volba velikost vzorku. Konkrétně u obrazu musí být velikost pixelu minimálně dvakrát menší než nejmenší detail, který je potřeba zachytit. Zobrazeni detailů znázorňuje obrázek 1.3.



Obrázek 1.3: Vzorkování obrazu [9].



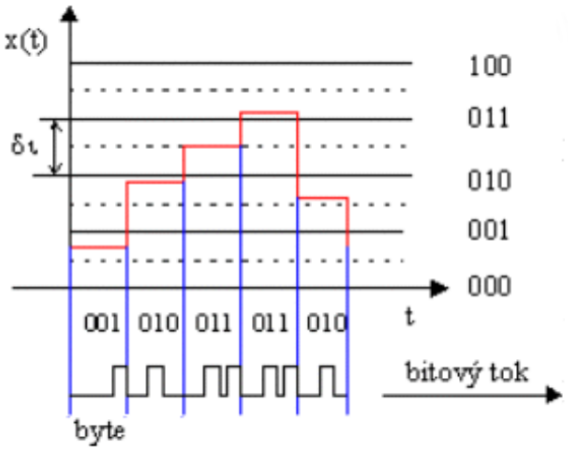
Originál 256×256 32×32

Obrázek 1.4: Snížení rozlišení [9].

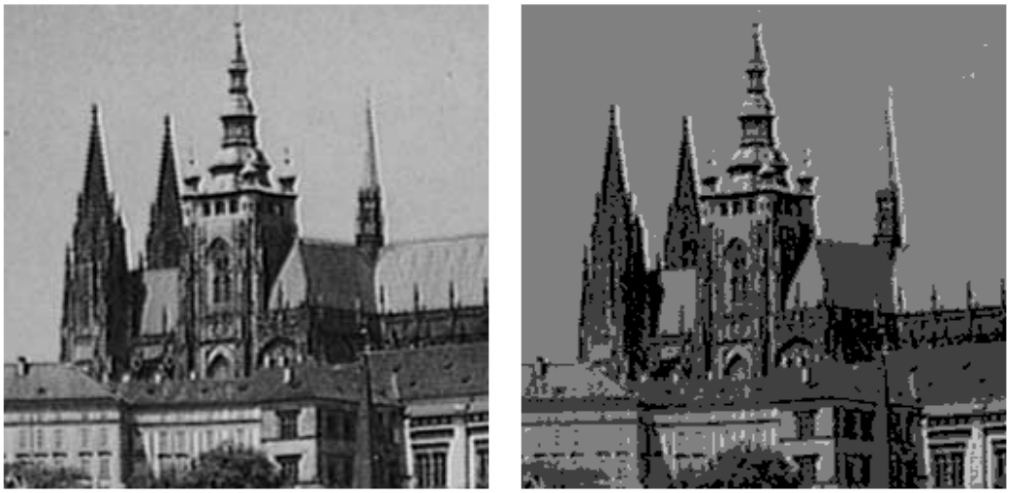
Je tedy důležité vhodně zvolit frekvenci a velikost rozlišení pro danou úlohu. Nemá smysl volit vysoké hodnoty, pokud to není nezbytné, protože s vyšším počtem snímků za sekundu a rostoucím rozlišením se i zvyšuje asymptotická složitost při zpracování obrazu.

Kvantování

Kvantování na rozdíl od vzorkování probíhá v oboru hodnot obrazové funkce. Je to tedy proces, který rozděluje funkci na intervaly, kterým je pak přidělena hodnota, nejčastěji jde o průměr celého intervalu. Kvantovaná funkce nemůže nabývat libovolné hodnoty, protože hodnota je zapsána ve dvojkové soustavě určité délky. Například pokud budeme uvažovat velikost jeden byte, získáme pouze 256 úrovní hodnot. Proto se vždy přiřazuje původní amplitudě nejbližší hodnota kvantovací úrovně [13]. To popisuje následující obrázky 1.5 a 1.6.



Obrázek 1.5: Kvantování [7].



Originál 256 jasových úrovní

4 jasové úrovně

Obrázek 1.6: Redukce jasové úrovně [9].

V každém obrázku se vyskytuje kvantizační chyba, která je způsobena přechodem mezi jednotlivými odstíny. Lidské oko tento jev zaznamená jako rušivé hrany, proto je dobré zvolit takové množství kvantizačních úrovní, aby daný algoritmus při zpracování obrázku nalezl to co hledá, zároveň ale je zbytečné volit co nejvíce úrovní, protože tím opět roste paměťová i výpočetní asymptotická složitost při zpracování obrazu či videa.

1.2.2 Barvy

Jednou z mnoha vlastností světla je barva, která vyjadřuje vlnovou délku daného fotonu. Lidské oko tedy zaznamená elektromagnetické záření a na základě vlnové délku mu přiřadí barvu. V lidském oku zaznamenávají barvu receptory zvané čípky, které jsou citlivé pouze na tři základní barvy: červenou, zelenou a modrou. Každý typ čípků je schopný změřit pouze určitý rozsah vlnové délky. Tabulka 1.2.2 popisuje viditelné spektrum jednotlivých barev:

Barva	Rozsah vlnových délek	Rozsah frekvencí
červená	625-800 nm	480-375 THz
oranžová	590-625 nm	510-480 THz
žlutá	565-590 nm	530-510 THz
zelena	520-526 nm	580-530 THz
modrá	430-500 nm	700-600 THz
fialová	400-430 nm	750-700 THz

Tabulka 1.1: Spektrum barev [15].

V počítačové grafice se barva tvoří na základě dostupných barevných modelů, což je vždy podprostor existujících barev, který slouží k rekonstrukci obrázku či scény. Nejpoužívanějším modelem, který se používá v moderních monitorech a projektorech je RGB model, který je založen na kombinaci tří barev (červená, modrá, zelená) různých odstínů. Výsledná barva tedy vzniká na základě intenzity jednotlivých barev. Intenzita či mohutnost jednotlivé základní barvy se určuje buď procentech, nebo podle barevné hloubky, která nabývá hodnoty podle počtu použitých bitů (nejčastěji se používá 8 bitů pro každou barvu). V praxi čím vyšší hodnotu nabývá 8-bitová komponenta pro danou barvu, tím intenzivněji svítí.

1.2.3 Předzpracování obrazu

V každém obrázku vyskytuje celá řada nedokonalostí a chyb. Každá úloha počítačového vidění vyžaduje určitý způsob předzpracování, aby se nakonec získal co nejlepší výsledek pro danou úlohu.

Potlačení šumu

Šum je v obrazu či videu přebytečná nežádoucí rušivá informace, která nejčastěji vzniká digitalizací či kompresí. Rozeznáváme dva typy šumů a to šum náhodný, který je způsoben nedokonalým senzorem. V obrázku pak můžeme vidět takzvané zrnění, což sou nežádoucí tečky, kde v těchto místech naměřil CCD element špatnou hodnotu neodpovídající realitě. Druhým typem šumu je Gausův šum, který je statistický a je popsán normálním (gausovským) rozdělením. Neexistuje způsob, jak

zcela odstranit jakýkoli šum, protože bychom musely znát jeho funkci což v praxi nikdy nemáme (můžeme mít pouze její aproximaci). Existuje celá řada metod, jak potlačit šum, bohužel většinou přitom ztratíme i část žádoucí informace jako jsou detaily či jas [18]. Nejpoužívanějšími metodami jsou:

Průměrování
Myšlenkou této operace je nahradit každý pixel v daném obrázku průměrem jistého okolí tohoto bodu. Velikost okolí je definováno konvoluční maskou, která zároveň popisuje, jak bude průměrování probíhat. Tomu se také říká vázané průměrování. Konvoluční masku si lze představit jako matici, která nabývá nezáporných hodnot, kde každá hodnota odpovídá danému pixelu v okolí. Hodnotu lze chápat jako váhu, tedy číslo, které určuje, jak je tento pixel důležitý. Pokud se zadefinuje veliké okolí, bude zřejmě, že pixely které leží na kraji tohoto okolí budou méně důležité, než ty ve středu. Součet hodnot konvoluční masky musí být jedna, aby nedošlo ke zesvětlení nebo ztmavnutí. Nevýhodou je, že čím je zvolena větší maska, tím více je potlačen šum, ale za cenu většího rozmazaní obrázku, proto je moudré volit masku vždy menší, než nejmenší detail, který je pozorován [18].

Gausovský filtr
Gausovský filtr je založen také na průměrování, ovšem liší se konvoluční maskou , kde jsou hodnoty dány Gasuovskou funkcí

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{1.1}$$

Tato metoda je nejefektivnější potlačení Gausovského šumu, bohužel opět platí čím větší okolí zvolíme, tím dojde k většímu rozmazaní, což může vést k celé řadě problému při dalším zpracování obrazu [18].

Mediánový filtr
Jak už název napovídá, Mediánový filtr je založený na hledání mediánu. Pro daný pixel se nalezne okolí, ze kterého se vybere medián, který se stává novou hodnotou pro zpracovávaný pixel. Narozdíl od předchozích metod patří Mediánový filtr do rodiny nelineárních filtrů a je vhodný na potlačení náhodného šumu [17].

Konzervativní vyhlazení
Velice rychlá metoda, která je založena na vyhledávání extrémů, tedy pixelů, který svou hodnotou vyčnívají. Stejně jako Mediánový filtr je tato metoda nelineární. Kolem zpracovávaného pixelu nalezneme okolí, ve kterém vyhledáme maximum a minimum. Pokud zpracovávaný pixel nabývá větší hodnoty než nalezené maximum, je mu přiřazeno právě nalezené maximum z jeho okolí. Pokud naopak je menší něž minimum, tak bude nabývat minima z jeho okolí. Výhodou je, že tato metoda je velice rychlá, protože má nízkou asymptotickou složitost [19].

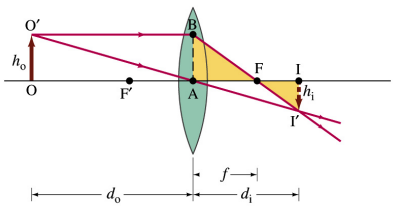
1.2.4 Kalibrace

Při natáčení videa či focení dochází k zobrazení 3D objektů dané scény na 2D plochu. Tomu se říká projekce. V dřívějších dobách první fotografický zařízení neobsahovala čočku, ale pouze malou dírku. Pokud probíhal proces focení, došlo k otevření oné malé dírky, která pustila malý svazek světla do komory, kde se promítl na fotografický papír a tím vznikaly první fotografie. Přestože se jedná o jednoduchý proces, používal se (byť trochu upravený) ještě mnoho let poté, než došlo k příchodu fotoaparátů s čočkou. Při projekci tehdejších primitivních fotoaparátů nedocházelo k takovému zkreslení jako dnes. Příčinnou není samotný fotoaparát či kamera, ale čočka, která zkresluje obraz.

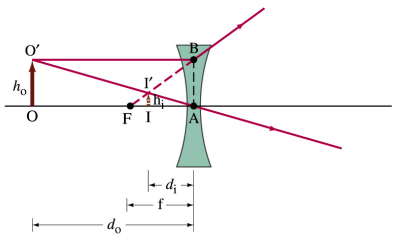
Optická čočka

Optická čočka je optická soustava dvou centrovaných ploch, nejčastěji kulových, nebo jedné kulové a jedné rovinné plochy. Čočka je tvořena průsvitným materiálem nejčastěji křemičitým sklem. Optická čočka se nejčastěji využívá v optice, protože ovlivňuje šíření světla a to jak viditelného, tak i infračerveného a ultrafialového. Mezi nejdůležitější fyzikální vlastnosti optické čočky patří index lomu, který je vždy větší než jedna a ovlivňuje rychlost šíření světla o různých vlnových délkách, dále je to absorpce daného materiálu (nejčastěji skla), která by se měla blížit co nejvíce nule, aby nedocházelo k absorpci světla čočkou. Nejdůležitějším parametrem optické čočky je ohnisková vzdálenost, což je vzdálenost optické čočky od jejího ohniska. Existuj dva druhy optických čoček a to spojky 1.2.4, které mají kladnou ohniskovou vzdálenost a rozptylky 1.2.4, které mají naopak zápornou ohniskovou vzdálenost. Výslednou změnu trajektorie světla při průchodu optickou čočkou matematicky popisuje zobrazovací rovnice dané optické soustavy [16].

$$\frac{1}{d_o} + \frac{1}{d_i} = \frac{1}{f}. \tag{1.2}$$

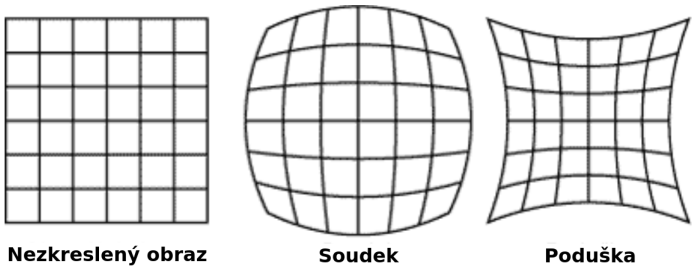


Obrázek 1.7: Spojka [14].



Obrázek 1.8: Rozptilka [14].

Nedokonalost optických čoček je způsobena mimo jiné nečistotou použitého materiálu. To se projeví na špatné průchodnosti světla (větší absorpce) a výsledný obraz nebude kvalitní. Druhým problémem je nedokonalost při obroušení, kdy povrch čočky neodpovídá dokonalé kulové ploše. Tím dochází ke sférickým vadám a to vede ke zkreslení obrazu.
Sférické vady jsou nejčastěji dvojího typu a to buď soudek nebo poduška jak je vidět v obrázku 1.9. To je typické hlavně pro širokoúhlé objektivy nebo pro zoom objektivy. V praxi to jsou parazitické jevy, které je nutné vykompenzovat.



Obrázek 1.9: Sférické vady [12].

Kalibrace radiálního zkreslení

Vady optických čoček zapříčiňují chyby, které se promítají do reálného obrazu lze vykompenzovat sadou koeficientů. Proces, který upravuje výstupní data z kamery se bude v následujícím textu nazývat korekce obrazu. Aby bylo možné správně upravit obraz a tedy aby celý proces probíhal korektně, je nutné nalézt odchylku pro každou část obrazu, neboť odchylka může být v každé části jiná.
Asi nejjednodušší způsob, jak zjistit chybu kamerové projekce je detekce referenčního předmětu v obraze, který je viditelný a následně porovnat zachycený obraz s referenčním předmětem ve skutečnosti. Poté se změří velikost i tvar referenčního předmětu na videu i ve skutečnosti. Velice vhodným referenčním předmětem je šachovnicové pole, které disponuje čtverci a přímkami.
Porovnání obrazu se skutečností může probíhat hledáním bodů, úseček či křivek. My budeme hledat body. V obraze se najde dobře viditelný bod, například roh čtverce a zjistí se jeho 2D souřadnice. Následně se snímek porovná s modelem či jiným kvalitnějším záznamem a odhadneme, kde by se měl daný bod nacházet na základě deformovaných přímek, které by byly při korektní kamerové projekci rovné. Tím se spárují dva body, které budou využity k nalezení projekční matice. Čím více se nalezne bodů, tím bude projekční matice přesnější. Je nutné hledat body v každé části obrazu a nezaměřit se jen na jednu část, byť jsou tam dobře viditelné. Došlo by k tak k takzvané lokální korekci, a byla by dobře potlačena projekční chyba jen v dané části obrazu a v jiných částech by mohla být chyba ještě vyšší, než bez korekce.

Proto je nutné hledat body rovnoměrně v každé části obrazu, čímž budou nejlépe odhadnuty parametry projekční matice.
Celý proces porovnávání může probíhat manuálně, poloautomaticky nebo plně automaticky. Manuální kalibrační proces je zcela závislý na lidském faktoru, protože kameru kalibruje pouze uživatel. Uživatel na základě dvou snímků, či své vlastní intuice vybírá body na obrazovce např. myší. Na základě těchto bodů se odhadnou distorzní koeficienty

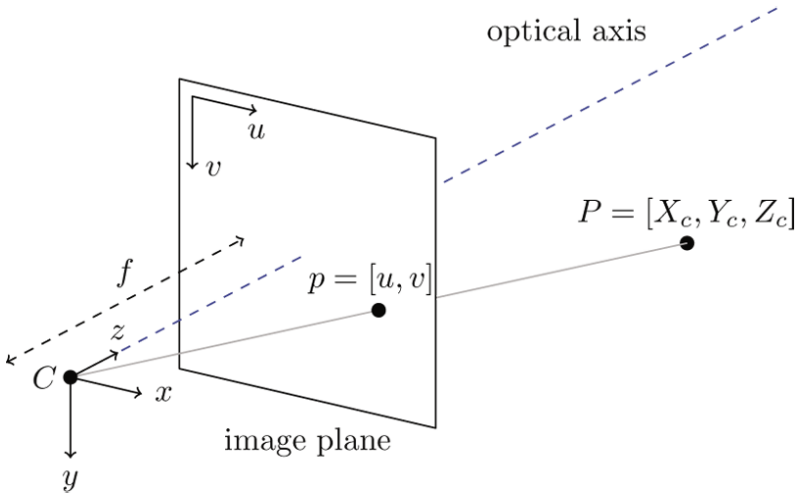
$$Dist_{coef} = \begin{pmatrix} k_1 & k_2 & p_1 & p_2 & k_3, \end{pmatrix} \tag{1.3}$$

kde koeficienty k_1, k_2, k_3 popisují radiální zkreslení a koeficienty p_1, p_2 tangenciální zkreslení, které bude zanedbáno.
Poloautomatická kalibrace je proces, při kterém se zařízení s kamerou kalibruje samo, ale je vyžadována pomoc např. od uživatele. Tato metoda vyžaduje referenční předmět, na kterém jsou dobře detekovatelné úsečky či kružnice. Příkladem může být šachovnicové pole. Při kalibraci touto metodou kamera snímá referenční předmět ve všech místech obrazové roviny, proto uživatel musí postupně přesouvat referenční předmět na všechna místa, které jsou v záběru kamery. Koeficienty kalibrační matice jsou vypočteny na základě deformace referenčního předmětu [4].
Plně automatický proces je nejsložitější. Kalibrace probíhá na základě kamerou detekovaných objektů v dané scéně, které jsou následně porovnány s databází objektů. Po nalezení schody je zjištěna deformace detekovaných předmětů a vypočteny správné pozice daných pixelů

$$x_{corrected} = x[1 + k_1 r^2 + k_2 r^4 + k_3 r^6] \qquad y_{corrected} = y[1 + k_1 r^2 + k_2 r^4 + k_3 r^6]. \tag{1.4}$$

Transformace

Kamerové projekce definuje vztah mezi obecným 3D bodem v prostoru a jeho 2D projekcí na obrázku či videu pomocí ideálního modelu kamery. Geometrické zobrazení 3D prostoru do 2D obrázku nazývá perspektivní promítání. Aby bylo možné matematické vyjádření perspektivního promítání 3D bodu do obrazové roviny, je nutné definovat několik důležitých pojmů a vztahů. Důležitějším bodem pří perspektivním promítání je střed, který se označuje C. V ideálním modelu přes tento bod procházejí všechny světelné paprsky, které dopadají na obrazový senzor kamery. Je to tedy optický střed a zároveň počátek souřadnicového systému kamery. Optickou osa kamery je přímka, která prochází optickým středem a zároveň je kolmá k obrazové rovině (obrazovému senzoru kamery). V následujícím obrázku se optická osa se značí z a ostatní osy se značí x a y. V počítačové grafice se většinou nachází nulový bod v levém horním rohu obrazovky. Optická osa z vyjadřuje hloubku. Pozici na obrazové rovinně znázorňují osy u a v.



Obrázek 1.10: Kamerová projekce [1].

Kalibrační matice zobrazí obecný bod $P = [X_c, Y_c, Z_c]$ ve 3D prostoru na bod $p = [u, v]$, který vznikne projekcí na obrazovou rovinu. Mezi body (X_c, Y_c, Z_c) a (u, v, f) je trojúhelníková podobnost, pro kterou platí:

$$\frac{u}{X_c} = \frac{v}{Y_c} = \frac{f}{Z_c}, \tag{1.5}$$

z tohoto vztahu je možné vyjádřit souřadnice na obrazové rovině:

$$u = \frac{fX_c}{Z_c} \qquad v = \frac{fY_c}{Z_c} \tag{1.6}$$

Rovnice budou převedeny do maticového tvaru:

$$\begin{pmatrix} u \\ v \\ z \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \tag{1.7}$$

To platí pro systém s počátkem nacházejícím se ve středu obrazové roviny. Počátek sořadnicového systému obrázku se nachází v levém horním rohu. Aby bylo možné počítat s tímto počátkem souřadnicového systému, je nezbytné upravit souřadnicové rovnice, ke kterým se připočítá velikost obrazové roviny:

$$p_u = fX_c + c_u \qquad p_v = fY_c + c_v \tag{1.8}$$

To platí pouze pro obrazovou rovinu, kde je poměr stran ekvivalentní. V praxi je většinou vertikální osa větší než horizontální, tím je poměr jiný, v dnešní době např. 9:16. To je možné vykompenzovat, pokud se v první rovnici přičte člen, který vyjadřuje poměr stran. Nejdůležitějším parametrem je α, který v případě rovnosti stran je nulový.

$$p_u = fX_c + \alpha f_x v + c_u \qquad p_v = fY_c + c_v \tag{1.9}$$

Rovnice je možné opět převést do maticového tvaru:

$$\begin{pmatrix} p_u \\ p_v \\ 1 \end{pmatrix} = \begin{pmatrix} f & \alpha f_x v & c_u \\ 0 & f & c_v \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \tag{1.10}$$

Pro kalibraci kamery většinu parametrů není nutné znát, protože matici lze vypočítat na základě přeučené soustavy homogenních lineárních rovnic. V sekci Kalibrace bylo popsáno několik způsobů, jak lze trénovací body získat. Z trénovacích bodů bude sestavena přeučená soustava homogenních lineárních rovnic, což znamená, že rovnic je mnohem více než neznámých kořenů. V případě kalibrace kamery je to jednodušší, protože neznáme 3D pozici bodu (musel by být součástí kamery 3D senzor).

1.2.5 Geometrické transformace

Geometrické transformace jsou nedílnou součástí počítačového vidění a počítačové grafiky. Hlavním účelem geometrických transformací je poskytování důležitých metod a funkcí pro manipulaci s objekty či celým obrázkem. To se využívá především v geometrickém modelování, kde se modeluje obraz z dat či převádíme snímek do jiného datového formátu. Například pokud obrázek obsahuje mnoho stejných či podobných objektů, stačí si zapamatovat jeden z nich a ostatní zobrazit na základě geometrické transformace. Tím může dojít k jisté ztrátě informace, ale ušetří se mnoho paměti. Především ve videu každý snímek je vetšinou podobný tomu předchozímu. Nejdůležitějšími geometrickými operacemi je posunutí, otočení, zkosení a změna měřítka. Výsledná geometrická transformace se pak skládá z kombinací těchto operací což umožní měnit pozici, tvar či velikost požadovaných objektů.

Translace

Posunutí bodu či objektu v rovině či prostoru je geometrická operace, která umožňuje změnít pozici bodu či objektu v souřadnicovém systému. Pokud se nacházíme v kartézském souřadnicovém systému, pak velikost a směr posunutí definuje translační vektor

$$T = \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{1.11}$$

Pozice nového bodu se získá přičtením translačního vektoru k původní pozici bodu P.

$$P' = P + T = \begin{pmatrix} x + t_x \\ y + t_y \end{pmatrix} \tag{1.12}$$

Po homogenizaci je získáma matice v tomto tvaru:

$$T_h = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \tag{1.13}$$

Rotace

Rotace je operace, která umožňuje pohybovat bodem či objektem po specifikované kružnici. Aby bylo možné provést rotaci, je nutné si specifikovat úhel rotace, což je úhel o který bude daný bod či objekt otočena druhým parametrem je střed rotace, což je bod, kolem kterého je rotace vykonávána. Dalším důležitým aspektem je točivý systém. Standardně se volí systém pravotočivý, kde kladný úhel rotace znázorňuje rotaci proti směru hodinových ručiček a záporný úhel naopak po směru hodinových ručiček. S touto skutečností v praxi bývá občas nesrovnalost, ale to platí především pro 3 prostor, ve 2 je to standardní záležitost. Ve 2D prostoru je rotace nejčastěji definována rotační maticí:

$$R = \begin{pmatrix} cos(\phi) & -sin(\phi) \\ sin(\phi) & cos(\phi) \end{pmatrix} \tag{1.14}$$

Zkosení

Zkosení je operace, která zkosí nebo nakloní objekt podle dané osy nebo pod určeným úhlem vzhledem k dané ose. Objekt či obraz se deformuje kolem zvoleného bodu a může probíhat ve všech třech rozměrech. Velikost zkosení pro určitý směr definují parametry Z_x, Z_y, Z_z. Obecné zkosení se většinou skládá z kombinací pro každý směr daného souřadnicového systému. Ve 3D prostoru obecné zkosení určuje 6 parametrů.

$$A_{xy} = \begin{pmatrix} 1 & 0 & Z_x \\ 0 & 1 & Z_y \\ 0 & 0 & 1 \end{pmatrix} \qquad A_{xz} = \begin{pmatrix} 1 & Z_x & 0 \\ 0 & 1 & 0 \\ 0 & Z_z & 1 \end{pmatrix} \qquad A_{yz} = \begin{pmatrix} 1 & 0 & 0 \\ Z_y & 1 & 0 \\ Z_z & 0 & 1 \end{pmatrix} \tag{1.15}$$

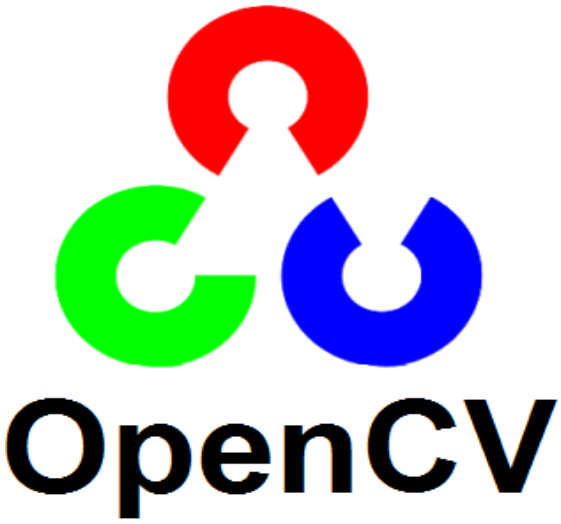
Změna měřítka

Tato transformace umožňuje měnit velikost obrazu či 3d prostoru pro danou souřadnicovou osu. Tím je umožněno roztahovat či smršťovat scénu. Scaling transformace se nejčastěji používá na upravování velikosti obrázků či videa, protože pokud má video příliš vysoké rozlišení, může následné zpravovaní trvat příliš mnoho času, proto je výhodnější upravit velikost a zmenšit rozlišení, aniž by došlo ke ztrátě důležitých informací. Důležité je proto zachovat poměry stran, tím dojde pouze ke ztrátě malých detailů, které se většinou ani nepozorují.

[noend]algpseudocode

1.3 OpenCV

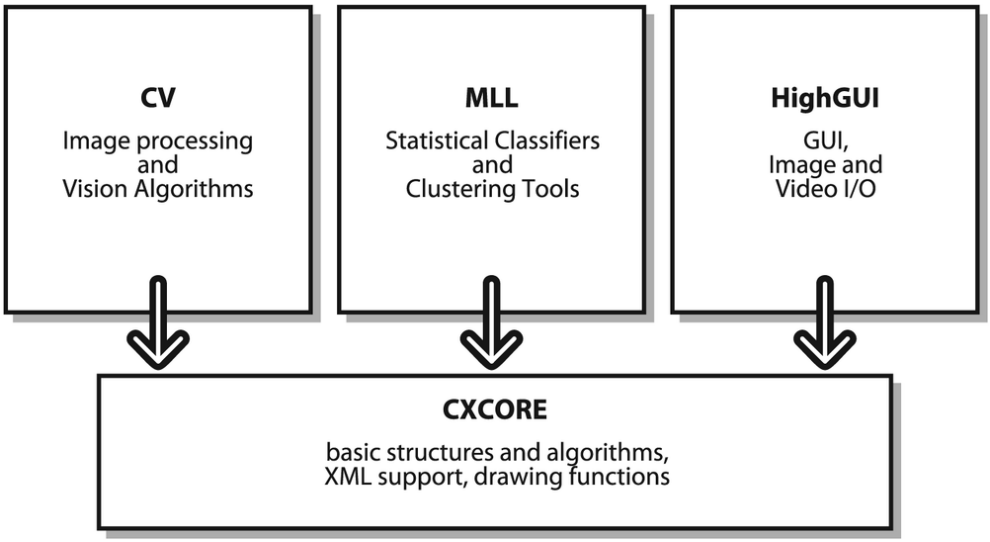
OpenCV je multiplatformní softwarová knihovna, která poskytuje mnoho užitečných funkcí a algoritmů pro práci a manipulaci s obrazem a videem. Využívá se hojně v oblastech počítačovém vidění a strojovém učení, pro které byla primárně vyvinuta společností Intel v roce 1999, dnes je volně šiřitelná pro výzkum, vývoj i komerční účely pod licencí BSD. Hlavním důvodem vývoje byl velký rozvoj aplikací počítačového vidění, v tehdejší době příliš zatěžovaly procesor a nebyly optimalizované. Cílem tedy bylo vytvořit multiplatformní knihovnu, která by usnadnila programátorům práci při vývoji složitých aplikací využívající zpracování obrazu.
Výhodou je, že mnoho algoritmů je v knihovněimplementováno jak pro CPU tak GPU, přičemž dnešní GPU má mnohem vyšší výpočetní výkon než CPU. Tyto algoritmy mohou byt využity pro detekci obličejů, identifikaci objektů, klasifikaci lidské činnosti ve videu, či extrakci 3D objektů ze scény. OpenCV má širokou komunitu čítající více než 47 000 členů.
Díky dispozici interface pro programovací C, C++, Java, Python, MATLAB a podpoře operačních systémů Windows, Linux, Android a Mac OS je OpenCV velice vhodnou knihovnou pro tuto práci [10].



Obrázek 1.11: Logo openCV [3].

1.3.1 Struktura

OpenCV je knihovna obsahující mnoho algoritmů a funkcí, které umožňují vyřešit širokou škálu problémů z oblasti umělé inteligence. Proto je možné rozdělit OpenCL knihovnu na několik částí z nichž každou je možné chápat jako samostatnou knihovnu, která poskytuje mnoho nástrojů zaměřených na daný obor. Obrázek 1.12 popisuje rozdělení do základních bloků:



Obrázek 1.12: Diagram struktury openCV [2].

CXcore

Základem celé knihovny OpenCV je blok CXCore, který propojuje všechny ostatní bloky. Je to řídící blok (knihovna), na které jsou ostatní bloky závisle, proto je CXCore základem každé aplikace využívající knihovnu OpenCV. CXcore obsahuje základní datové typy, lineární algebru obsahující maticové operace a transformace, nebo algoritmy zajišťující správu paměti.
Nejdůležitějším datovým typem, který se používá téměř pro každou reprezentaci dat, je struktura CvMat, která může reprezentovat matici všech typů, obrázek či vektor. Jedná se o velice užitečný nástroj pro reprezentaci matic v jazyku C/C++, protože tato struktura umožňuje reprezentovat matice různých typů. Dále umožňuje provádět důležité maticové operace jako je např. sčítání, násobení či maticová inverze. Veškeré výpočty jsou optimalizované. Většina dalších struktur a tříd je odvozena od cvMat např. IplImage, která je obohacená o některé prvky práci s obrázky.
Získání přímo číselných nebo jiných dat z matice či struktuře od ní odvozené je velice využívaná operace, proto existuje celá řada metod. Jazyk C/C++ umožňuje

přímý přístup do struktury, což je asi nejefektivnější způsob získání dat. Dále je možné využít maker, ale nejvíce uživatelsky přátelsky a také nejvíce využívaný je pomocí get a set funkcí. Proto mnoho struktur obsahuje vnitřní funkce, které počítají s maticemi.
V knihovně CXCore je implementováno mnoho funkcí, které umožňují vykreslování celé řady geometrických objektů (úsečka, čtverec, kruh či text). Při vykreslování obrázků se ve většině případů využívá struktura cvMat či její odvozené struktury. Hlavním důvodem je, že obrázek je reprezentován jako matice $m \times n$ (m řádků a n sloupků), kde každá buňka matice reprezentuje daný pixel, který může patřit do modelu RGB či reprezentovat odstín šedi (pokud de o černobílí obrázek). Dále je umožněno vykreslení textu přímo na obrazovku či do matice. To je realizováno fonty, které jsou součástí Cxcore [10].

CV

Knihovna CV poskytuje mnoho algoritmů a funkci, které se využívají v počítačovém vidění a ke zpracování obrazu. Patří sem funkce pro detekce hran, různé lineární transformace obrazu, morfologie či nástroje pro tvorbu histogramů. Dále knihovna CV poskytuje funkce pro strukturální analýzu obrazu, optický tok či rozpoznávání různých vzorků z obrazu.
Obrázky v počítači mohou být reprezentovány vektorově nebo diskrétně. Vektorové jsou tvořeny vektory, který popisují dané objekty v obrázku. To představuje velikou výhodu, protože vektorový obrázek zabírá mnohem méně místa na úložném zařízení. Při transformaci změna měřítka nedochází ke ztrátě informace. Většinu obrázků tvoří diskrétní obrázky, které jsou reprezentovány jako matice pixelů. V knihovně CV se rozlišují dva tipy obrázku a to binární a mnohoúrovňové. Binární obrázek je nejjednodušším reprezentace snímku, protože v každém bodě může nabývat pouze dvou hodnot. Obrázek, který může být v daném místě černý nebo bíly. Tento tip obrázku se využívá při zobrazení hran testovaného obrázku, či zobrazení rozdílu mezi dvěma obrázky. Druhým tipem je mnohaúrovňový obrázek, což je klasický model obrázku, kde každý pixel může nabývat několika různých hodnot. Typickým příkladem je RGB snímek, kde každý pixel může nabývat 256 různých hodnot. Také využít model černobílého obrázku, kde každý pixel může nabývat úrovně šedi. To se využívá především v obrázcích, kde specifická barva nereprezentuje důležitou informaci jako je například vyhledávání objektů v dané scéně. Pokud je převeden obrázek z RGB do černo-bílého modelu, dojde sice ke ztrátě určité informace, výsledné zpracování bude mnohem rychlejší [10].

HighGUI

OpenCV program běžící na daném zařízení musí komunikovat s grafickým prostředím, aby zobrazil např. video na monitor. Komunikaci zprostředkovává knihovna HighGUI. Jedná se o typické operace které zajišťuje operační systém jako například

tvorba oken, obrázků či jiné grafické operace. Operace pracující s úložným prostorem, tedy ukládání a načítání dat na disk a práce soubory, kde je uložené video. Veškeré funkce, které knihovna HighGUI nabízí jsou vhodné pro GUI (Graphical User Interface). Pro složitější grafické aplikace je výhodnější použít jiné grafické knihovny např. GTK. Knihovnu HighGUI je možné rozdělit na dvě části, kde jedna část zajišťuje načítaní, ukládání dat a veškeré operace s tím spojené, druhá část poskytuje nástroje na ovládání GUI.
Načítání a ukládání dat je standardní operace, kterou zajišťuje operační systém. Aplikace OpenCV nejčastěji načítá obrázkové či video soubory a kalibrační data ze speciálních *xml* souborů. Knihovna HighGUI podporuje mnoho grafických formátů (JPEG, PNG, BMP a mnoho dalších), které je možno načíst ze souboru, nebo je naopak uložit do souboru v požadovaném formátu. Zdrojem obrázků, které jsou zpracovávány v OpenCV může být sekvence obrázků uložená na disku, video či stream z kamery nebo internetu. Práce s videem je závislá na platformě, pro kterou je OpenCV aplikace zkompilována, protože každý systém využívá jiný software (např. Linux FFmpeg).
Druhá část knihovny HighGUI je vývojáři označovaná jako Simple GUI. Podle funkcionality lze tuto část knihovny chápat jako mezivrstvu mezi aplikací OpenCV a operačním systémem. Hlavní myšlenkou je poskytnutí jednoduchých uživatelských funkcí pro práci s GUI bez nutnosti implementace složitých algoritmů, které jsou s tím spojeny. Typickým příkladem je zobrazení obrázku do vytvořeného okna definované velikosti. Dále knihovna HighGUI poskytuje vstupně/výstupní operace, jako je možnost přečíst vstup z klávesnice, myši či touchpadu [10].

MLL

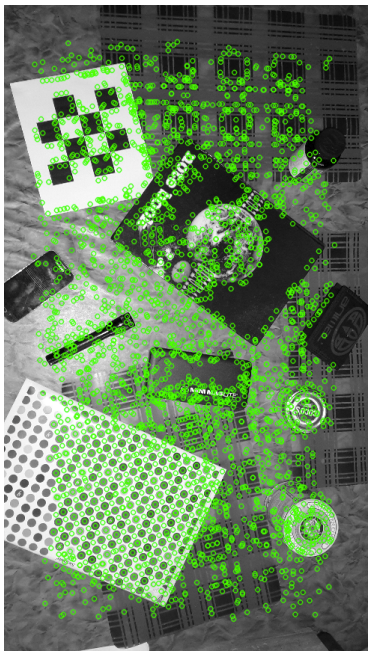
Knihovna MML přímo nesouvisí s počítačovým viděním, ale poskytuje metody obecnému strojovému učení, obor umělé inteligence, který se zabývá zpracováním dat a jejich klasifikací. Příkladem je problém, kde se klasifikátor naučí podle jedné sady dat a otestuje pomocí jiné sady dat. Tomuto procesu se říká predikce. Nejjednodušším klasifikátorem je hledání nejbližšího souseda, vyhledán kandidát z trénovací množiny na základě nejmenší odchylky. U vybraného kandidáta se zjistí do jaké skupiny patří a na základě této skutečnosti se přiřadí příznak testovacímu kandidátovy. V knihovně MLL jsou implementovány i tyto metody: SVM, Decision trees, Bayes či neuronové sítě.
Veškeré klasifikátory je možné využít i v počítačovém vidění. Typickým příkladem možnosti jejich využití je klasifikace objektů v obrázku. Nejjednodušším způsobem jak naučit klasifikátor detekovat určitý objekt je načíst ze souboru několik trénovacích objektů, které transformovat hranovým detektorem a následně naučit zvolený klasifikátor. Dalším krokem je detekce přímo hledaného objektu v obrázku. To je možné provést hrubou sílou, kde opět zvolený obrázek bude transformován hranovým detektorem, následně se musí prohledat každá část po části, aby byla 100% jistota, že se nevynechá žádné místo, kde by se hledaný objekt mohl nacházet [10].

1.3.2 Detekce příznaků obrázku

Obrázek je množina pixelů, kde každý nabývá určité hodnoty a lze je považovat za nositele informace, protože se nachází na určitém místě a nabývá jisté hodnoty. To je důležité při hledání zajimavých míst v obrázku, kde v daném místě pixeli tvorí určitou strukturu, které je unikátní. Pro zpracování či porovnání dvou obrázků by bylo výpočetně náročné pracovat se nezměněnou formou obrázku, protože při vysokých rozlišení může obrázek obsahovat milióny bodů (pixelů). Proto je výhodnější obrázek převést na množinu informací, které nejlépe charakterizují daný obrázek. Důvodem je eliminace zbytečných informací, které jsou nepodstatné a negativně ovlivňují výpočetní rychlost. V každém obrázku se nacházejí místa, které nesou více informace. Příkladem může být fotografie nebe, kde místa s mraky jsou mnohem zajímavější (pro popis) než místa s modrým pozadím. Obrázek tedy nejlépe popisují místa, kde dochází k určitým změnám. Tyto body se v anglickém jazyce nazývají features points (dále už jen příznaky) a slouží k popisu obrázku. Existuje mnoho metod jak příznaky najít a každá má své výhody a nevýhody. Cílem je najít co nejrychleji dostatečné množství příznaků, které dobře charakterizují obrázek.
Jak už bylo napsáno dříve obrázek tvoří množina pixelů. Většina metod vyhledávající příznaky mají společný základ. Obrázek si lze rozdělit na několik subobrázků. Lze si to představit jako puzzle, kde daný obrázek je rozřezán na několik dílků, kde každý dílek nese informaci o poloze (místo kam patří) a druhou informací je samotný obrázek na daném dílku. Podobný proces probíhá i při extrakci, kdy je z hlavního obrázku vybrán daný dílek (subobrázek) o specifické velikosti, která se definuje při inicializaci daného detektoru příznaků. Celý proces lze chápat jako dvojitý cyklus, kde proměnné x,y definují střed, který se v každém dalším kroku posune o specifickou vzdálenost, která je rovněž definována při inicializaci detektoru příznaků. Kolem každého středu je vždy vyříznut subobrázek specifické velikosti, který zpracuje daný detektor příznaků. Každý detektorzpracuje subobrázek svým specifickým algoritmem a zjistí jak je zajímavý. Pokud je zajímavý hodně (přesáhne se treshold), je přidán (jeho jeho souřadnice) do pole příznaků, které vrátí daný detektor [10]. Srovnání nalezených bodů pro jednotlivé detektory je vidět v obrázcích: 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19, 1.20 a 1.21.
Po nalezení příznaků (zajímavých bodů) aplikace disponuje pouze kartézskými souřadnicemi těchto bodů. To už přináší důležitou informaci o daném obrázku, ale ještě výhodnější je příznaky popsat číselnou hodnotou, která musí co nejlépe popsat nalezené body.
Proces extrakce příznaků (Příznakový extrahtor) je proces, při kterém se zajímavý subobrázek, který je reprezentován daným příznakem převede na číselný vektor. Ten charakterizuje obrazovou informaci kolem daného příznaku číselnou hodnotou. Extrakce vektoru je náročný proces jako hledání příznaků a existuje opět spousta metod, jak získat reprezentující vektor. Nalezený vektor je vždy normalizovaný, protože při porovnávání dvou vektorů (hledání schody) se počítá odchylka a ta v případě normalizovaných vektorů nikdy nepřesáhne hodnotu 1.

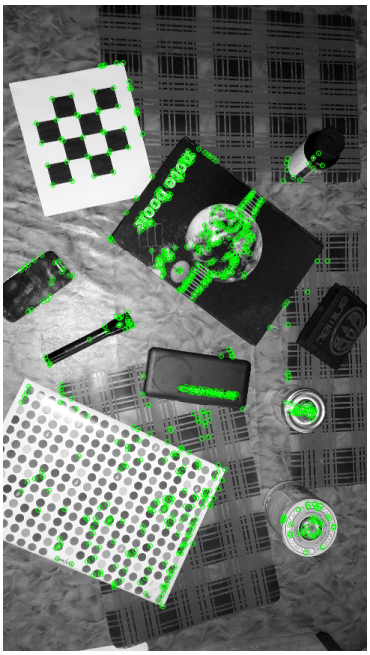
Typy Detektorů

- SURF (Speeded-Up Robust Features)
- SIFT (Scale-Invariant Feature Transform)
- Harris Corner Detection
- FAST Algorithm for Corner Detection
- BRISK (Binary Robust Invariant Scalable Keypoints)
- ORB (Oriented FAST and Rotated BRIEF)
- STAR
- GFTT (Good Features To Track)
- MSER (Maximally stable extremal region)

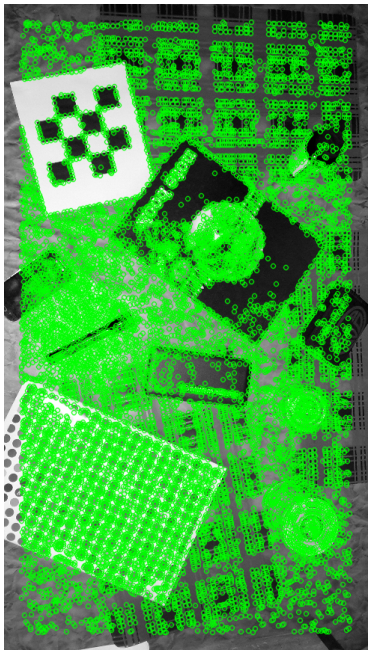


Obrázek 1.13: SURF

Obrázek 1.14: SIFT



Obrázek 1.15: Harris



Obrázek 1.16: FAST

Obrázek 1.17: BRISK

Obrázek 1.18: ORB

Obrázek 1.19: STAR

Obrázek 1.20: GFTT

Obrázek 1.21: MSER

1.3.3 Budoucnost

V dnešní době se veškeré podobory, patřící do počítačového viděný velice rychle rozvíjí. Každým rokem vzniká veliké množství aplikací, které plní funkci např. kontroly kvality výrobků (optic control), medicínské aplikace či pro navigaci na Marsu. Knihovna OpenCV je dlouhou dobu podporována společností Intel a v dnešní době získala podporu i od Willow Garage výzkumného institutu robotiky, což znamená že knihovna OpenCV se bude v příštích letech hodně rozvíjet a snad pronikat do více a více aplikací.
Jednou z nových oblastí na kterou je v dnešní době při rozvoji knihovny OpenCV kladen zvlášť veliký důraz je robotické vnímání jehož součástí je 3D vnímání, což je stejné jako pozorování 2D a 3D objektů. Robotické vnímání spoléhá na dobré 3D snímání okolního prostředí, proto se vyvíjí metody pro korespondenci systému s několika kamery či 3D senzory (3D lidar). V dnešní době je na trhu mnoho 3D senzorů či infrakamer, proto je kladen důraz na vývoj pokročílých metod kalibrace, které umožňují začlenit tyto komponenti do snímacího aparátu. Rozpoznávací systémy sisponující 3D senzorem mají obrovskou výhodu, protože výpočet hloubky z klasické kamery je velice obtížné [10].

1.3.4 Ukázkový kód aplikace v OpenCV

```
#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace cv;

int main(int argc, char** argv )
{
    VideoCapture cap;
    if ( argc != 2 )
    {
        cap = VideoCapture(0); // otevreni vchoz webkamery
    }
    else
    {
        cap = VideoCapture(argv[1]); // nacteni videa ze souboru, ktery je urcen
            argumentem
    }

    if(!cap.isOpened()) // kontrola zdroje dat
    {
        return -1;
    }

    Mat edges;
    namedWindow("edges",1);
    for(;;)
    {
        Mat frame;
        cap >> frame; // ziskani ramce z kamery
        cvtColor(frame, edges, CV_BGR2GRAY); //prevedeni do cerno-bileho obrazku
        GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5); //Smoothing obrazku
        Canny(edges, edges, 0, 30, 3); //nalezeni hran
        imshow("edges", edges); //vykresleni obrazku do GUI
        if(waitKey(30) >= 0) break; //zjisteni konce videa
    }
    // the camera will be deinitialized automatically in VideoCapture destructor
    return 0;
}
```

1.4 OpenCL

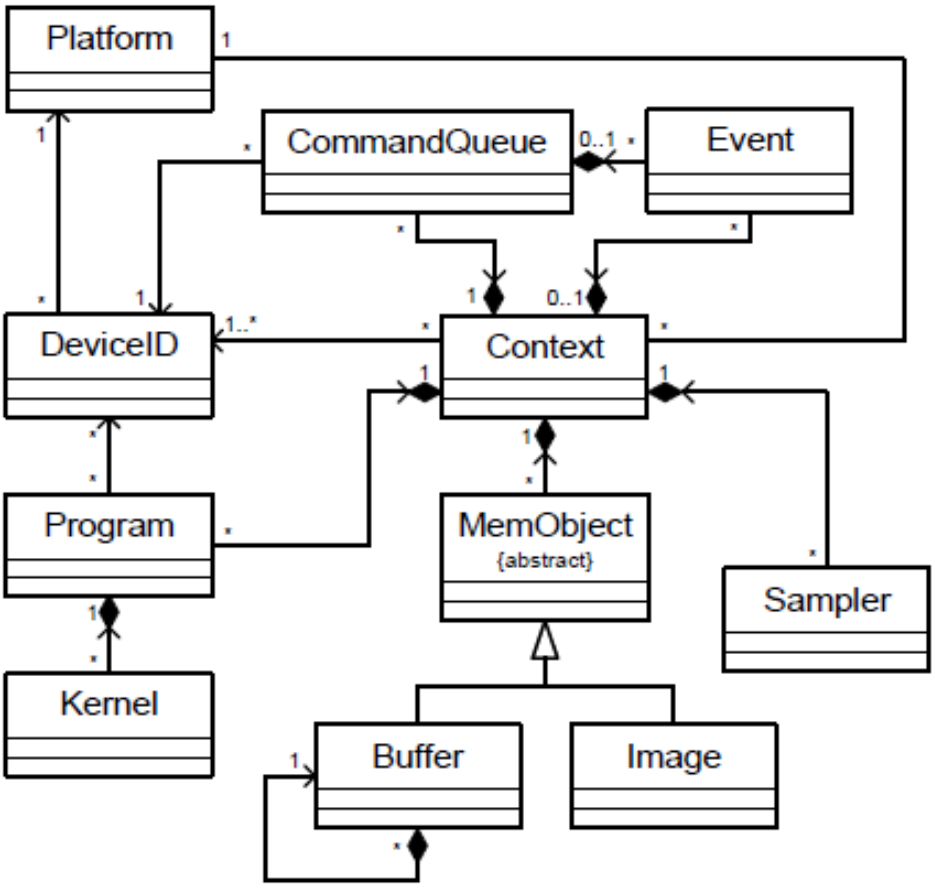
OpenCL (Open Computing Language) je standard pro programování výpočetních systému, které disponují GPU (graphic processing unit), APU (accelerated processing unit) nebo DSP (digital signal processor). V dnešní době je obtížné zvyšovat pracovní frekvence procesorů či GPU. Dnešním trendem je navyšování počtu jader výpočetních systémů. To se týká především procesorů na grafických kartách, které v dnešní době disponují tisícovkami výpočetních elementů. To poskytuje obrovský výpočetní výkon, proto při vývoji moderních aplikací je kladen důraz na paralelizaci. Moderní Operační systémy poskytují synchronizační nástroje na základě mutexů a semaforů. Na grafické kartě žádný operační systém neběží a jednotlivé výpočetní jádra jsou mnohem jednodušší než u klasického CPU. Proto na grafické kartě synchronizační nástroje neexistují a programování je mnohem těžší.

1.4.1 Architektura

Otevřený standard OpenCL se skládá z modelů: Platformní model, Paměťový model, Vykonávací model, Programovací model

Platformní model

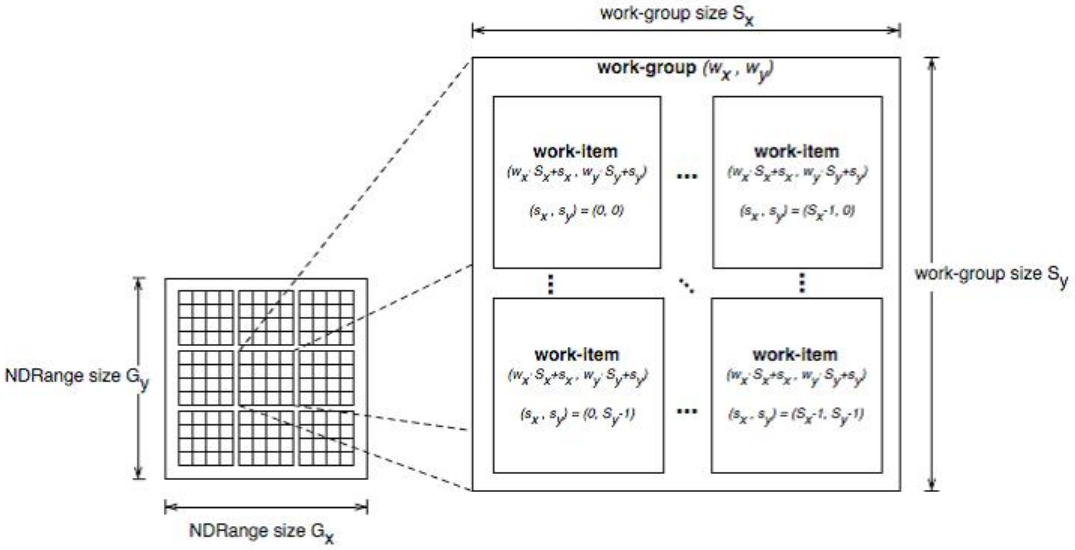
Platformní model je možné chápat jako výpočetní stroj či počítačový systém, který je schopný nabízet a podporovat OpenCL. Platformní model se skládá z z hostitelského systému, který dále může obsahovat jedno nebo více zařízení OpenCL. Typickým příkladem je osobní počítač, který obsahuje kromě procesoru (CPU) i grafickou kartu (GPU), kde obě výpočetní jednotky jsou odlišné architektury, ale obě podporují výpočty standardu OpenCL. Každé zařízení může obsahovat několik výpočetních jednotek, které dále mohou obsahovat několik prvků pro zpracování dat. Dnešní grafické karty obsahují tisíce výpočetních prvků, což značně převyšuje počet jader dnešních CPU.
Aplikace OpenCL se vykonává na hostitelském systému jako klasický program, který poté odesílá příkazy, které inicializují vykonávání výpočtů na výpočetních prvcích v daném zařízení. Každý prvek je možné chápat jako samostatný výpočetní procesor, který vykonává instrukce. Zásadní rozdíl oproti klasickému procesoru je skutečnost, že výpočetní prvek může přistupovat pouze do místa paměti, který mu je přidělen. To platí především pro ukládání dat, Protože GPU postrádá synchronizační prvky jako jsou např. mutexy či semafory [8].



Obrázek 1.22: Diagram struktury OpenCL [8].

Exekuční model

Běh openCL programu probíhá ve dvou částech: aplikační část a kernel modul. Aplikační část odpovídá za spouštění a komunikaci kernel procesů na daném zařízení, které jsou vyjádřeny jazykem OpenCL. Celý program se skládá z jednoho nebo více vláken, které jsou vykonávány na daném zařízení. Než je spuštěn samotný kernel na výpočetní jednotce, je nutné definovat indexovaný prostor (který může být až třírozměrný), počet instancí kernelu a velikost skupin do kterých se sdružují instance. Spuštěná instance, která je vykonávána na výpočetním prvku se nazývá pracovní položka (work-item) a vždy je ji přidělen unikátní identifikační číslo podle indexovaného prostoru.
Pracovní položky dále mohou být sdružovány do pracovních skupin, kterým je stejně jako pracovním položkám přiděleno unikátní skupinové identifikační číslo. Pracovní skupina poskytuje možnost komunikace pracovních položek mezi sebou. Komunikace probíhá přes sdílenou paměť a mohou ji využívat pouze pracovní položky, které jsou sdruženy v jedné skupině. Indexový prostor se nazývá NDRange a může nabývat až tří rozměrů. Pracovní položka po inicializaci v argumentu získá jeden až tři indexy, které definují její paměťový
prostor. To samé se vykonává i u pracovních skupin, kde každé skupině jsou přiřazeny identifikátory podobným způsobem jako u pracovních položek. Pracovní položky jsou přiřazovány do skupin na základě svého ID, které se indexuje od nuly. Ve skupině je pracovní položka identifikována na základě ID pracovní položky a ID skupiny. Kombinace těchto dvou prvků musí být vždy unikátní [8].



Obrázek 1.23: Architektura exekučních jednotek na čipu [8].

Paměťový model

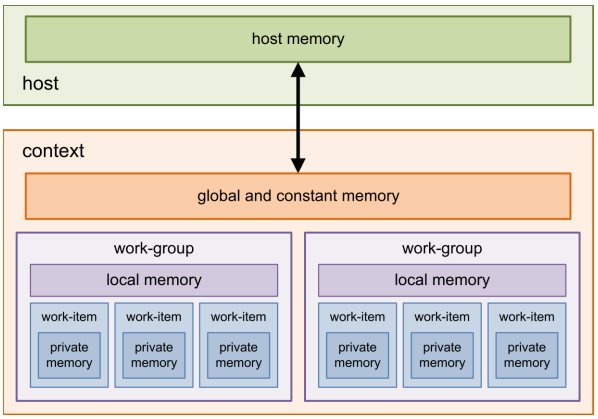
Paměťový model znázorňuje paměťovou hierarchii, která specifikuje typy a druhy přístupu. Existují 4 druhy pamětí do kterých mají pracovní položky přístup:

Globální paměť umožňuje čtení i zápis všem pracovním jednotkám z libovolné skupiny. Pracovní jednotky mohou přistupovat a zapisovat data do libovolné části Globálního paměťového objektu. Globální paměť je viditelná všem pracovním položkám, proto při zápisu je nutné

Konstantní paměť je jako Globální paměť s tím rozdílem, že zůstává konstantní po celou dobu běhu kernelu. Aplikace tuto paměť inicializuje na začátku běhu kernelu, do které jsou uložena data. Jednotlivé pracovní položky mohou z této paměti pouze číst.

Lokální paměť je určena pracovním skupinám, které ji využívají při alokování proměnných. Do lokální paměti mohou přistupovat pouze pracovní jednotky z dané skupiny, pro které je tato paměť určena. Lokální paměť platí stejné přístupové práva jako pro globální paměť.

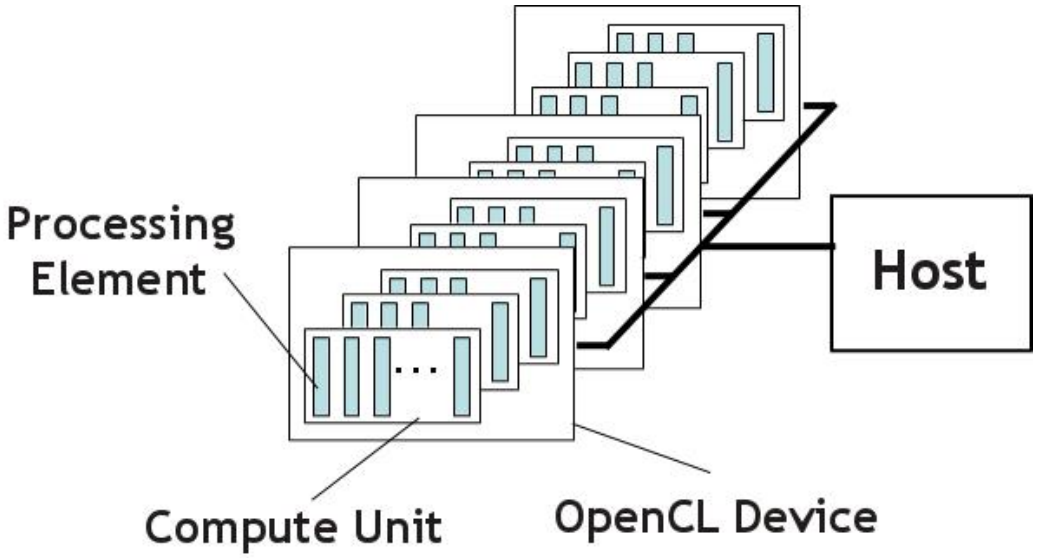
Privátní paměť je určena pouze pro jednotlivé pracovní jednotky. Do této kategorie se řadí veškeré definované proměnné, které jsou určeny pouze pro danou pracovní jednotku. Ostatní pracovní jednotky k této paměti nemají přístup.



Obrázek 1.24: Paměťové schéma [8].

Programovací model

Standard OpenCL je navržen pro podporu úlohově paralelních a datově paralelních programovacích modelů či hybridní model, který vzniká jejich kombinací. Hlavní důraz je kladen na datově paralelní programovací model, který definuje výpočet jako souběh instancí kernelu (pracovních položek), které zpracovávají datové položky. Veškeré datové položky jsou jednoznačně namapované indexovacího prostoru z exekučního modelu. Většinou připadá jedna datová složka pro jednu vykonávací jednotku, ovšem může tomu být i jinak. OpenCL poskytuje hierarchický datový model, který se dělí na explicitní a implicitní. V explicitním modelu je nutné specifikovat celkový počet pracovních položek (běžících současně) a velikost pracovních skupin, v implicitním modelu pouze počet pracovních položek, protože rozdělení do skupin zajišťuje OpenCL.
Úlohově paralelní programovací model je možné chápat jako multitasking, protože umožňuje souběžné vykonávání několika různých kernelů. Neumožňuje však souběh více instancí stejného kernelu (pokud nejde o hybridní model), což je hlavní rozdíl oproti datovému paralelnímu modelu. Úlohově paralelní programovací model je možné přirovnat k aplikačnímu modelu na procesoru, kde každý proces je možné chápat jako jeden kernel [8].



Obrázek 1.25: Schéa zařízení s jednotkami OpenCL [8].

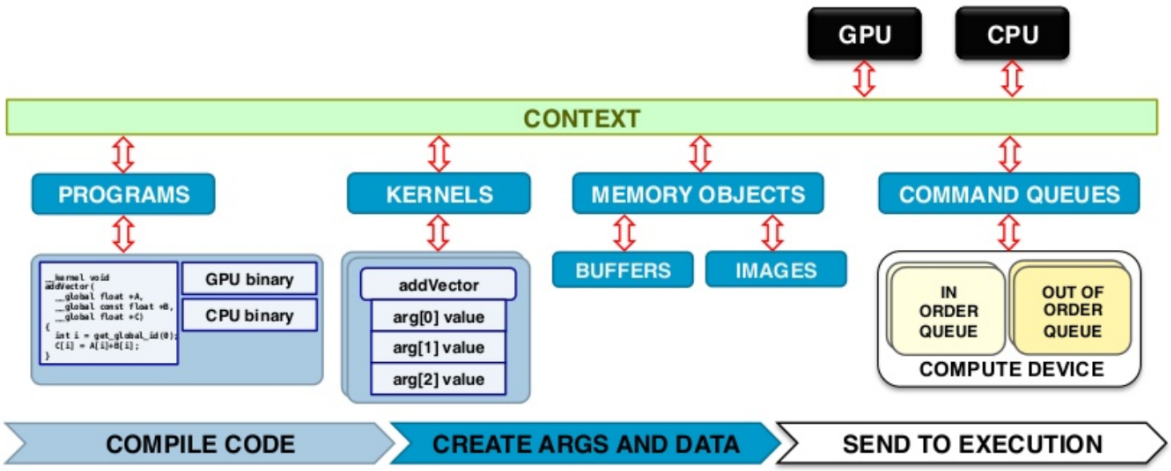
1.4.2 Framework OpenCL

OpenCL framework poskytuje aplikacím možnost využívat hostitelský systém pro vykonávání výpočtů. Systém se může skládat z více OpenCL zařízení, které mohou aplikace OpenCL frameworku využít současně. OpenCL framework se skládá z následujících komponent:

- **Platformní vrstva**
  Platformní vrstva prozkoumá a následně poskytuje informace o daném zařízení, na kterém může běžet OpenCL. Uživatel se může dotázat platformní vrstvy na počet OpenCL zařízení na daném systému, počet stream procesorů GPU, architekturu GPU a na mnoho dalších důležitých informací. Některé informace mohou být následně předány kompilátoru, který je může využít pro výběr OpenCL zařízení, na kterém bude kernel sestaven a vykonán.

- **Kompilátor**
  Kompilátor je vrstva, která zmapuje specifické zařízení. Po předání zdrojového kódu jádra kompilátor sestaven obraz, který je spuštěn v zařízení. Výhodou tohoto procesu přenositelnost OpenCL aplikace na jiné zařízení bez nutnosti rekompilace, protože kompilace jádra probíhá během runtime procesu.

- **Runtime**
  Runtime vrstva se využívá při spuštění zkompilovaných jader dané aplikace, kde poskytuje několik důležitých funkcí pro správu paměti a přenos dat na dané zařízení.



Obrázek 1.26: Model rámce OpenCL [8].

1.4.3 Ukázkový kód

Zdrojový kód OpenCL je možné rozdělit na dvě části a to aplikační část a kernel. Aplikační část inicializuje OpenCL zařízení ve kterém alokuje paměť či do zařízení zkopíruje data, se kterými bude proveden výpočet. V První části zdrojového kódu aplikační část přijímá tři matice. První dvě matice (a,b) budou vynásobeny a výsledek bude zapsán do matice "c".

```
/*aplikacni cast*/
void gpucompute(int *a, int *b, int *c, int rowsA, int colsA, int rowsB, int
    colsB)
{
    /*podminka pro nasobeni matic*/
    if(colsA != rowsB)return;

    cl::Buffer d_a, d_b, d_c;
    cl::Context context(DEVICE);
    cl::Program program(context, util::loadProgram("pathtokernel/kernel.cl"),
        true);
    cl::CommandQueue queue(context);
    auto vadd = cl::make_kernel<cl::Buffer, cl::Buffer, cl::Buffer, int, int,
        int>(program, "nasobeni_matic");

    /*transfer dat do pameti GPU*/
    d_a = cl::Buffer(context, &a[0], &v_a[rowsA*colsA-1], true);
    d_b = cl::Buffer(context, &b[0], &v_b[rowsB*colsB-1], true);
    d_c = cl::Buffer(context, CL_MEM_WRITE_ONLY, sizeof(int) * rowsA * colsB);

    /*spusteni kernelu*/
    vadd(
         cl::EnqueueArgs(
            queue,
            cl::NDRange(rowsA * colsB)), //pocet procesu
            d_a, d_b, d_c, manyrows, manycols, count
         );

    queue.finish();

    /*zkopirovni vysledku do Ram pameti*/
    cl::copy(queue, d_c, &c[0], &c[rowsA * colsB-1]);
}
```

Kernel je část kódu, která běží na OpenCL zařízení (GPU). Pro jednoduchou představu je možné chápat tento úsek kódu jako funkci, která se zavolá tolikrát, kolik bude mít členů výsledná matice. Funkce může být spuštěna několikrát paralelně, podle počtu jader, které obsahuje dané zařízení openCL.

```
/*kernel*/
__kernel void matrix_multiplication(
    __global int* A,
    __global int* B,
    __global int* R,
    const int rowsA, const int colsA,
    const int rowsB, const int colsB)
{
    int index = get_global_id(0);
    int rowC = index / rowsA;
    int colC = index % colsB;
    int sum = 0;
    int i;
    for(i = 0;i < colsA;i++){
        sum += A[rowC*colsA + i] * B[colC + colsA*i];
    }
    C[index] = sum;
}
```

Kapitola 2

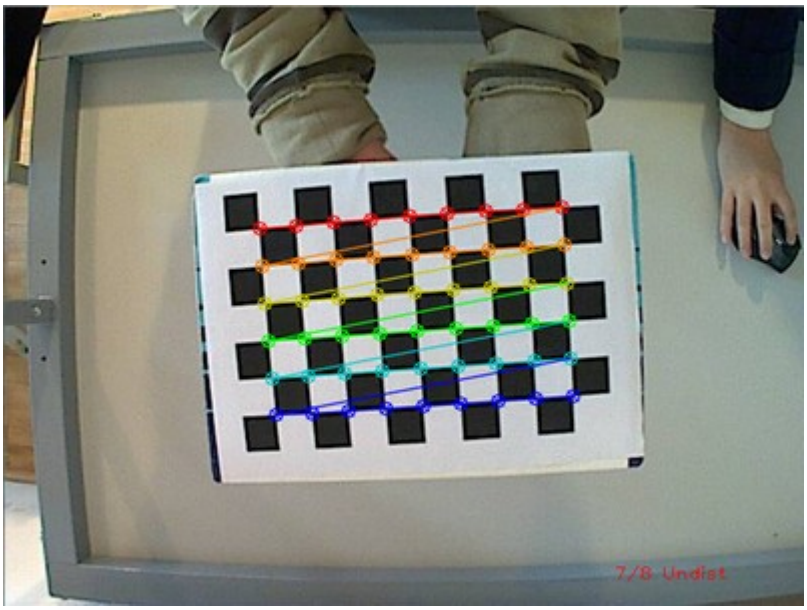
Implementační část

2.1 Kalibrace

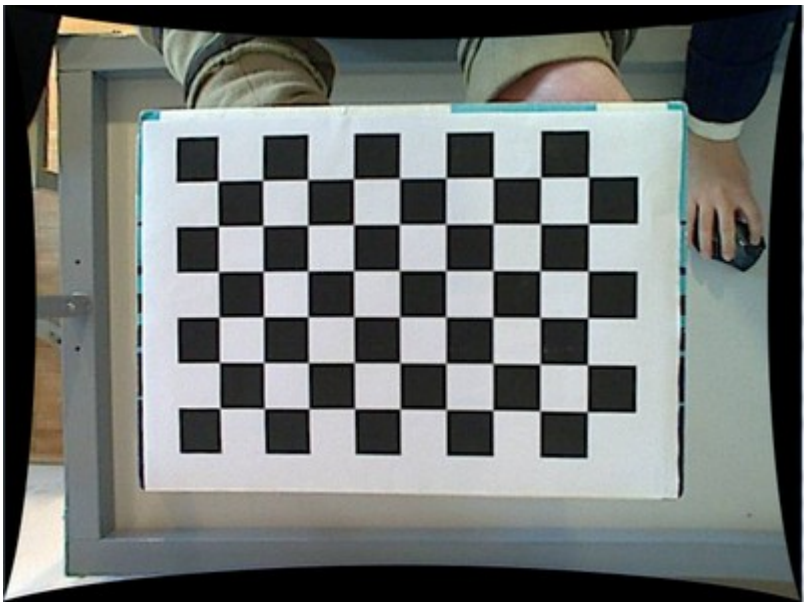
V této práci kalibrace kamery představuje důležitou operaci, pokud čočka na kameře produkuje velkou odchylku v daném místě. Kalibrace není podstatná, pokud drona disponuje dobrou kamerou, která nezkresluje obraz. V předchozích částech této práce byl popsán teoretický postup, jak získat parametry pro kalibraci. V praxi s využitím knihovny openCV se zavolá sekvence funkcí, které spustí kalibraci. Bude se jednat o poloautomatický proces. Pro kalibraci kamery bude využitý referenční objekt šachovnice. Při procesu kalibrace knihovna openCV bude vyhledávat šachovnicové pole a detekovat přímky. Protože se jedná o poloautomatický proces, je nutné, aby s kalibrací pomáhal člověk. Uživatel spustí funkci runCalibrationAndSave, která načte video nebo sekvenci snímků podle nihž bude kalibrace probíhat. Snímky obsahují šachové pole v různých pozicích. Výsledné koeficienty budou zapsány do XML/YAML souboru. Výsledkem této operace je:

- **určení distorzních koeficientů**
- **určení kalibrační matice**
- **načtení vstupu z kamery, videa nebo množiny obrázků**
- **načtení konfigurace z XML/YAML souboru**
- **uložení výsledku do XML/YAML souboru**
- **vypočítání re-projekční chyby**

[noend]algpseudocode [latin1]inputenc



Obrázek 2.1: Zkreslený obraz [5].



Obrázek 2.2: Zkreslený obrázek po korekci [5].

2.2 Korespondence snímků

Video či stream z kamery je sekvence snímků jdoucích po sobě za určitou periodu. Standardní frekvence klasického videa bývá kolem 20-30 snímků za sekundu. Přestože se nejedná o vysokou frekvenci, změny mezi jednotlivými snímky nejsou příliš veliké. Proto v této práci bude hledaná korespondence dvou snímku nepřímo jdoucích po sobě, ale s prodlevou jedné sekundy. Po tomto časovém úseku by měli být změny dobře viditelné a detekovatelné. Dalším důvodem je vysoká výpočetní náročnost hledání korespondence dvou snímků, která by nebyla vypočítána v tak rychlém čase pro každou dvojici po sobě jdoucích snímků. Pro hledání korespondence dvou snímků byly implementovány dvě metody: odchylková a příznaková metoda.

2.2.1 Odchylková metoda

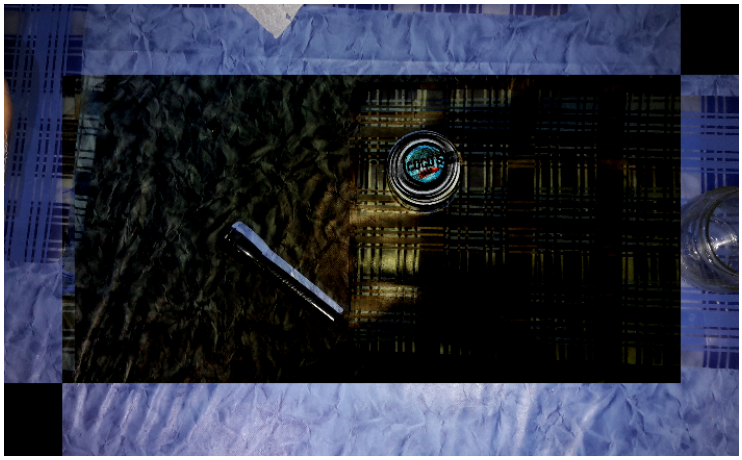
Odchylková metoda je založena na hledání minimální průměrné odchylky dvou obrázků. Obrázek je množina pixelů nabývajicích určité hodnoty. Proto je možné od sebe dva obrázky odečíst, kde výsledek bude ve formě absolutní hodnoty rozdílu dvou obrázků. Z toho je možné získat průměrný rozdíl pro každý pixel což je hledaná průměrná odchylka dvou obrázků. Tento výpočet je nutné vykonat několikrát, protože jeden obrázek se musí vůči druhému obrázku v každém kroku posunout v ose x či y. Průměrná odchylka se vypočítá jen v místě, kde se překrývají. Tato metoda se dá klasifikovat jako korespondence hrubou silou, protože pro každou pozici se musí vykonat výpočet. To přináší značné problémy, hlavně ve vysoké výpočetní náročnosti této operace. Další velkou nevýhodou je nemožnost detekce rotace, zkosení či změna měřítka. Proto tato metoda je vhodná pro video kde dochází pouze k posunu kamery v ose x a y, což je znázorněno v obrázcích: 2.3, 2.4 a 2.5.

Scénář 1: Sesazení snímků hledáním nejmenší průměrné odchylky.

Data: *imageA, imageB*
Initialization: $\theta_{best} \leftarrow \infty$;
$x_{best}, y_{best} \leftarrow 0$;
for $x \in \{-x_{max}, \ldots, x_{max}\}$ **do**
  for $y \in \{-y_{max}, \ldots, y_{max}\}$ **do**
    imageA' = moveImage(*imageA*, *x*, *y*)
    θ = getAverageDistance($imageA', imageB$)
    if $\theta < \theta_{best}$ **then**
      $\theta_{best} \leftarrow \theta$
      $x_{best}, y_{best} \leftarrow x, y$
    end
  end
end
Return: $\theta_{best}, x_{best}, y_{best}$

Obrázek 2.3: Původní snímek.

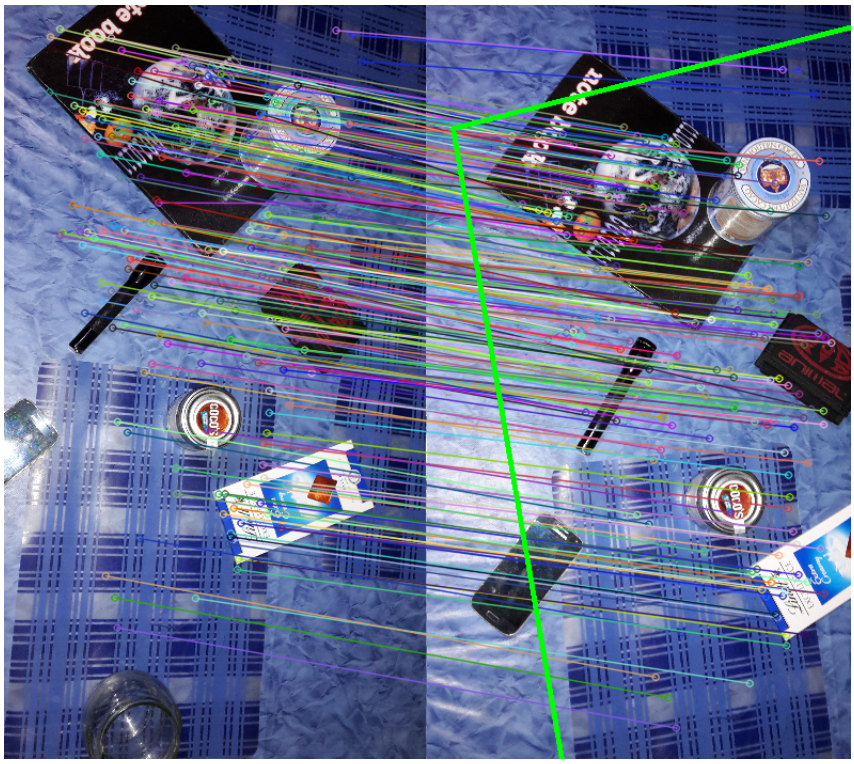
Obrázek 2.4: Posunutý snímek.



Obrázek 2.5: Korespondence snímků včetně zbytku.

2.2.2 Příznaková metoda

Příznaková metoda je založena na hledání zajímavých bodů u po sobě jdoucích snímků. Obecně neplatí pravidlo čím více bude nalezeno bodů, tím více bude transformace přesnější, protože jednotlivé body je nutné spárovat. To je operace, která příznaku z obrázku A přiřadí příznak z Obrázku B. Proto je výhodné vypočítat u každého příznaku deskriptor což je normalizovaný číselný vektor, který charakterizuje daný příznak. Pro daný bod z obrázku A se vybere nejpodobnější příznak na základě nejmenší odchylky deskriptoru.
Po spárování bodů je mnoho bodů chybně spárováno, proto bude vybráno pouze 30% bodů s nejmenší odchylkou. Tyto bodu budou využity při hledání transformace mezi dvěma obrázky. Pro tento účel je možné využít mnoho algoritmů např. ICP (Iterative closest point). V této práci bude využita metoda warp, která je součástí knihovny OpenCV. Metoda warp má velikou výhodu, protože je schopná detekovat posunutí, rotaci, zkosení i změnu měřítka.
Pro dosažení nejlepších výsledků je nutné pro danou scénu zvolit dobrý detektor příznaků. Problémem metody warp může být různá chybovost v dané části obrazu. Důvodem je různá hustota příznaků v každé části obrazu. Nejlepších výsledků bude dosaženo pouze s rovnoměrně rozprostřenými příznaky po celé ploše obrázků (2.6).



Obrázek 2.6: Spárováné příznaky dvou snímků.

Obrázek 2.6 znázorňuje dva snímky, kde v levé části je snímek A, v pravé části snímek B. U obou snímků byly vyhledány příznaky, které byly spárovány. Z celé množiny spárovaných příznaků bylo vybráno 10% nejlepších párů (s nejmenší odchylkou) nalezené páry příznaků, ze kterých byla funkcí warp vypočtena transformace. Zelený rámeček na obrázku B popisuje část obrázku kterou obsahuje i snímek A.

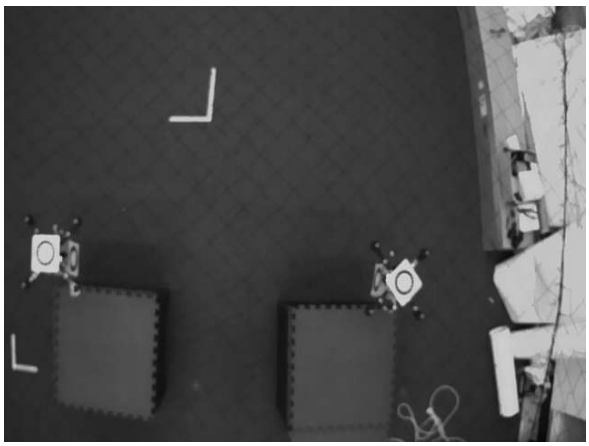
Obrázek 2.7: Transformace druhého snímku.

[noend]algpseudocode algpseudocode [latin1]inputenc [section]placeins amsmath amsfonts amssymb

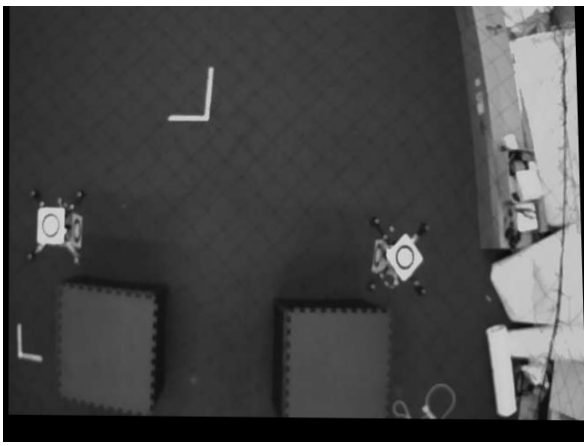
2.3 Detekce změn

2.3.1 Hledání definovaných objektu

Tato metoda je založena na hledání objektů určitého tvaru. Příkladem může být vyhledávání aut či lidí. K tomuto úkolu je nutný Objektový detektor, který je součástí knihovny OpenCV. Detektor je při inicializaci nutné naučit vyhledávat určitý tip objektů. K tomu slouží trénovací množina dat, např. snímky několika aut z různých úhlů. Další metodou jak naučit objektový detektor je načtení *xml* souboru, který obsahuje přímo číselné hodnoty popisů hledaného objektu.
Tato metoda není příliš univerzální, což je zároveň její výhodou i nevýhodou. Největší nevýhodou je poměrně vysoká výpočetní náročnost. V této práci nebude tato metoda ani testována, protože nalezené objekty u obou obrázků je nutné spárovat, což dělá příznaková metoda pro univerzální objekty.
Obě metody budou porovnány na následujících dvou snímcích:



Obrázek 2.8: Detekce změn: snímek A.

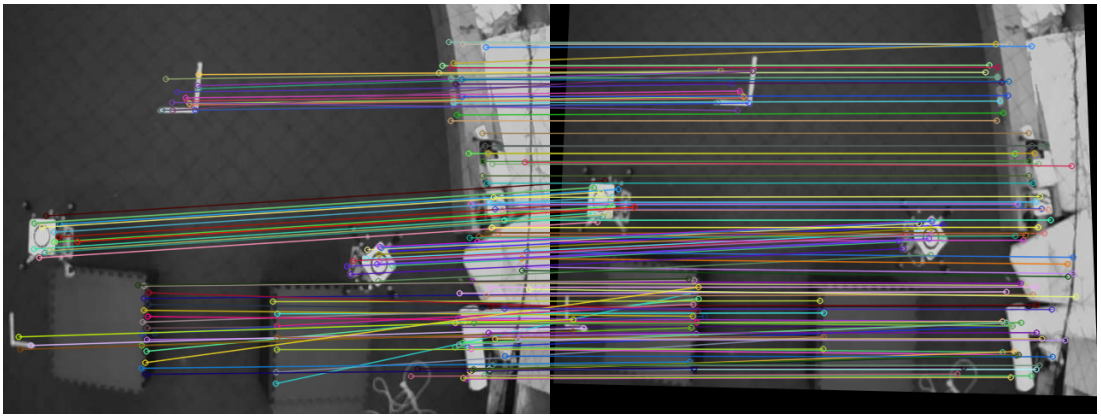


Obrázek 2.9: Detekce změn: snímek B.

2.3.2 Příznaková analýza

Jak už název napovídá, tato metoda detekce změn je založena na vyhledání příznaků pohybujících se objektů. Metoda musí nalézt dostatečné množství příznaků u obou obrázků tak, aby se příznaky nacházeli i na hledaných objektech. Proto je výhodnější nastavit proměnou *hesianTreshold* (proměnná, která ovlivňuje množství nalezených příznaků) na nižší hodnotu, aby bylo nalezeno více příznaků. Nesmí být snížena natolik, aby nedošlo k vysokému zpomalení aplikace, protože rychlost je ovlivněna množstvím dat. Nalezené příznaky je nutné spárovat pro dva po sobě jdoucí snímky.
V dalším kroku dojde k vymazaní přebytečných příznaků na základě vzdálenosti.

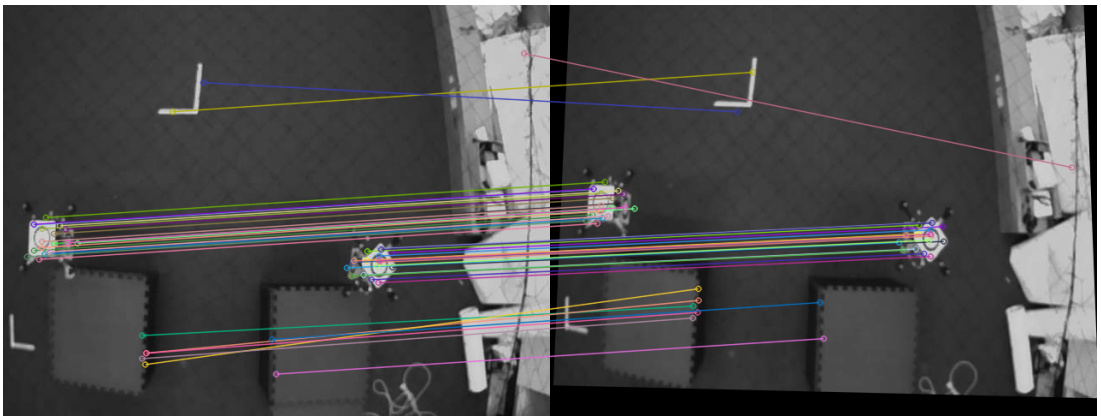
To je provedeno na základě hledaného vzdálenostního pásma, což je předpokládaná velikost posunu objektu. Absolutní vzdálenost spárovaných příznaků se musí nacházet v určitém intervalu, který je definován uživatelem. Tím se vymaže většina spárovaných příznaků.
Posledním a zároveň finálním krokem je vyhledání objektů na základě příznaků. Tato Operace musí příznaky rozdělit do skupin, kde každý objekt popisuje množina příznaků. Zároveň je nezbytné vyfiltrovat chybné páry příznaků, kterých je v této fázi ještě veliké množství. Na začátku algoritmu se vybere první příznak a dojde k vytvoření objektu. Následně dojde k vyhledání dalších příznaků, které do daného objektu patří. Tyto příznaky musí splňovat dvě podmínky: Příznak se musí se nacházet poblíž nějakého příznaku daného objektu (vzdálenost musí být menší než maximální definovaná mez). Druhá podmínka je založena na stejné závislosti vzdáleností příznaků u obou obrázků. Daný spárovaný příznak v daném objektu se musí nacházet ve stejné vzdálenosti od každého dalšího příznaku z dané skupiny pro oba obrázky. Toto pravidlo se dá chápat jako geometrická struktura. Pokud se vytvoří polygon z jedné skupiny příznaků patřící do stejného objektu, musí být tento polygon stejný u obou obrázků (stejný tvar).
Tato metoda je poměrně rychlá a univerzální. Nevýhodou je nemožnost detekce velice malých objektů, protože se příznaky na těchto malých objektech většinou nenacházejí. Dalším problémem může být změna pozadí, protože pokud se pohybující objekt přesune na jiné pozadí, příznakový deskriptor bude vypadat jinak a příznaky se nespárují. Důvodem je skutečnost, že příznaky se většinou nacházejí na hranách objektů.



Obrázek 2.10: 30% příznaků s nejmenší odchylkou.

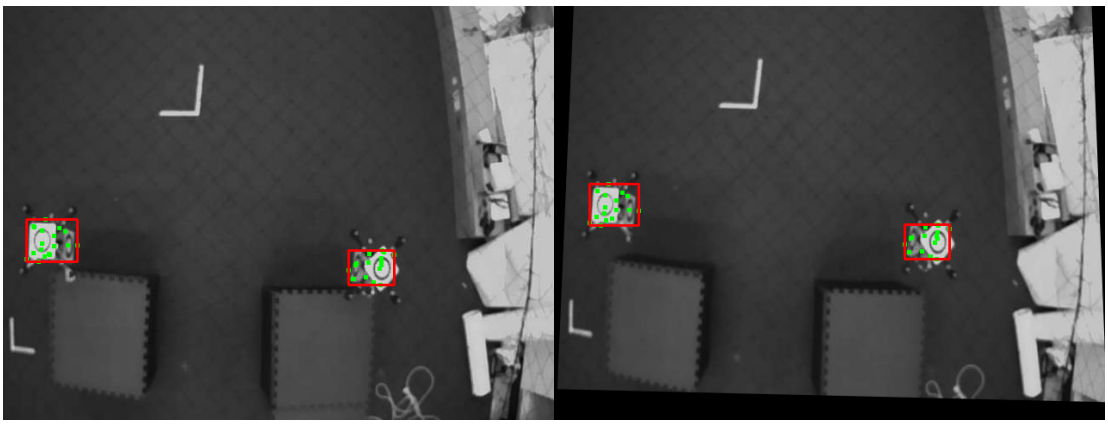
Obrázek 2.10 zobrazuje 30% spárovaných bodů s nejmenší odchylkou. Mnoho bodů se nachází i na pohybujících se objektech. V dalším kroku se musí vyfiltrovat body, které se nacházejí na pozadí. To bude vykonáno jednoduchým mezovým filtrem. Při inicializaci mezového filtru se definují dvě proměnné: $minDist$ a $maxDist$. Tyto proměnné definují o jakou minimální a maximální vzdálenost se hledaný pohy-

bující objekt může posunout. Poté je vypočten rozdíl jak moc jsou vůči sobě body jednotlivých spárovaných příznaků posunuté. Pokud se hodnota nachází v definovaném úseku, bude spárovaný příznak ponechán. V opačném případě bude vymazán.



Obrázek 2.11: Příznaky náležící definovanému intervalu.

Jak je na obrázku 2.11 vidět, téměř veškeré body, které se nenacházejí na pohyblivém objektu, byly vyfiltrovány. Všechny body se vyfiltrovat nepodařilo, proto je nutné v dalším kroku body přiřadit do shluku. V tomto případě se nejedná o klasický K-mean clustering, ale spíše o spektrální clustering rozšířený o jedno pravidlo. Existují dvě podmínky, zda bude další bod přidán do daného shluku. 1. Musí se nacházet poblíž některého bodu v daném shluku. 2. Musí se nacházet ve stejné vzdálenosti v obou snímcích od každého jiného bodu v daném shluku První pravidlo popisuje klasický spektrální clustering. Druhé pravidlo říká, že body musí v obou snímcích tvořit stejnou geometrickou strukturu.



Obrázek 2.12: Detekované pohyblivé objekty.

2.3.3 Zbytková analýza

Zbytková analýza je založena na odečtení dvou obrázků. Tato metoda vyžaduje dobře skorespondované snímky, protože už i minimální odchylka může vyprodukovat výraznou chybu. Vstupem jsou dva snímky, které se odečtou metodou absolutní odchylky pro každý pixel. Zbytek bude transformován a pokud hodnota zbytku přesáhne definovanou mez, bude dané místo vyhodnoceno jako změnové (kde dochází k pohybu). Body je pak nutné uspořádat do clusterů např. pomocí spektrálního clusteringu, což je metoda, která shlukuje body na základě vzdálenosti.

Obrázek 2.13: Zbytek po odečtení obrázků.

Na obrazku 2.13 je vidět zbytek po odečtení obrázků 2.8 a 2.9 v inverzní formě (invertované barvy). Zbytek je zřetelný převážně u pohyblivých objektů (helikoptéry), ale i od jiných předmětů. Důvodem je nedokonalá korespondence snímků. Bohužel ani dokonalá korespondence neeliminuje chybové zbytky, protože pokud se v dané scéně nachází vysoký objekt (např. skříň), tak zobrazení tohoto objektu na kamerovou rovinu bude jiné.
Existuje možnost jak převést množinu snímků daného prostředí na 3D model, avšak to je velice obtížné a mimo rámec této práce. Jednodušší metodou je modifikace odečítacího algoritmu. Ten lze modifikovat tak, že odečítání nebude probíhat přímo mezi jednotlivými body se stejnou polohou, ale v prvním obrázku se vybere pixel z

daného okolí vybraného bodu a z tohoto okolí se vybere pixel, který je nejpodobnější vůči pixelu z druhého obrázku. Tyto pixely pak budou odečteny.
Výhodou modifikovaného odečítacího algoritmu je eliminace chybových zbytků, nevýhodou naopak zmenšení rozdílu (hledaný zbytek nebude tak výrazný). Proto je nutné vhodně zvolit mez okolí což je hodnota, která definuje velikost okolí ve kterém se vybírá nejpodobnější bod. Dalším problémem je vysoká výpočetní náročnost, protože výrazně vzroste asymptotická složitost toho algoritmu na $O(4kmn)$, kde m je počet řádků, n počet sloupců a k je velikost okolí. Už i pro obrázek o velikosti 640×480 výpočet trvá několik minut na běžném procesoru. Proto je nutné vyžít grafickou kartu na které se výpočetní čas sníží na několik desetin sekundy.

Scénář 2: Modifikovaný odečítací algoritmus.

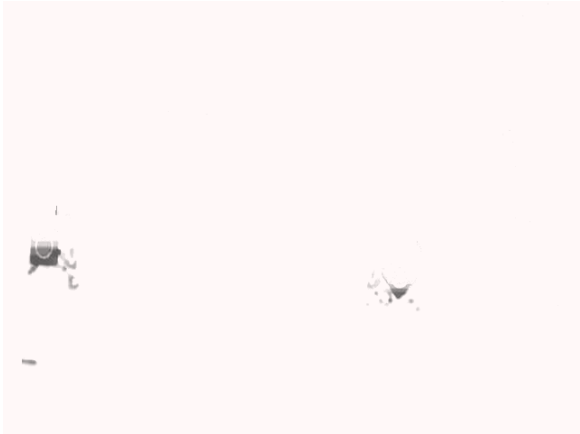
```
   Data: imageA, imageB
1  Initialization : resImage ← newImage(imageB.rows, imageB.cols);
2  begin
3  |  for x ∈ {0, ..., imageB.cols − 1} do
4  |  |  for y ∈ {0, ..., imageB.rows − 1} do
5  |  |  |  ω = getMostSimilarPixel(imageA, x, y, treshold, imageB.pixel[x][y])
   |  |  |  resImage.pixel[x][y] = imageB.pixel[x][y] - ω
6  |  |  end
7  |  end
8  end
9  Return: resImage
```

figureZbytek pro: $mez = 3$.

figureZbytek pro: $mez = 7$.

float table placeins

figureZbytek pro: $mez = 10$.



figureZbytek pro: $mez = 15$.

Kapitola 3

Experimenty

Experimenty budou vykonány na nahrávkách natočených z 4-vrtulových helikoptér (kvadrikoptéra). Na videu je natočena scéna s pohybujícíma se objekty. Videa byla rozkouskována na sekvenci po sobě jdoucích snímků s prodlevou do 1 sekundy.

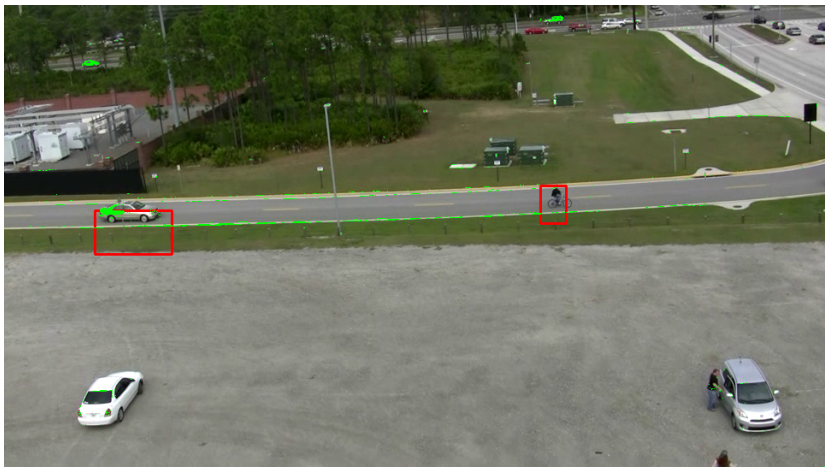
3.1 Experiment č. 1

U tohoto experimentu bude analyzováno video, které bylo převzato ze stránek univerzity: . Video bylo natočeno kvalitní zkalibrovanou kamerou, proto bude vynechán krok kalibrace kamery. Dále pak nedochází k velkým vibracím, proto se na videu téměř nenachází žádný šum. Na videu je možné pozorovat pohyblivé objekty: projíždějící auta, cyklisté, chodící lidi a běžící lidi. Délka videa je 1 minuta 41 sekund, frekvence snímků je 24 Hz a rozlišení je 960×540. Pro detekci pohybujících se objektů budou spuštěny obě implementované metody (příznaková analýza a zbytková analýza), kde detekované objekty příznakovou analýzou budou zakresleny do červených rámečků (obrázek 3.1, 3.4 a 3.5) a rozdíly, které detekuje zbytková analýza budou vykresleny zeleně (obrázek 3.9).
Pro zajištění korespondence snímků byla využita příznaková metoda pomocí SURF příznaků, kde byl zvolem $hesianTreshold = 100$ a pro funkci warp bylo vybráno 30% spárovaných příznaků s nejmenší odchylkou.
Pro Příznakový detektor pohybu byl zvolen $hesianTreshold = 200$ a bylo vybráno 40% spárovaných příznaků s nejmenší odchylkou. Pro detekci pohybujících se bodů byl zvolen interval $\langle 20, 150 \rangle$ pixelů.
Pro Zbytkový detektor byl zvolen $distanceTreshold = 7$.

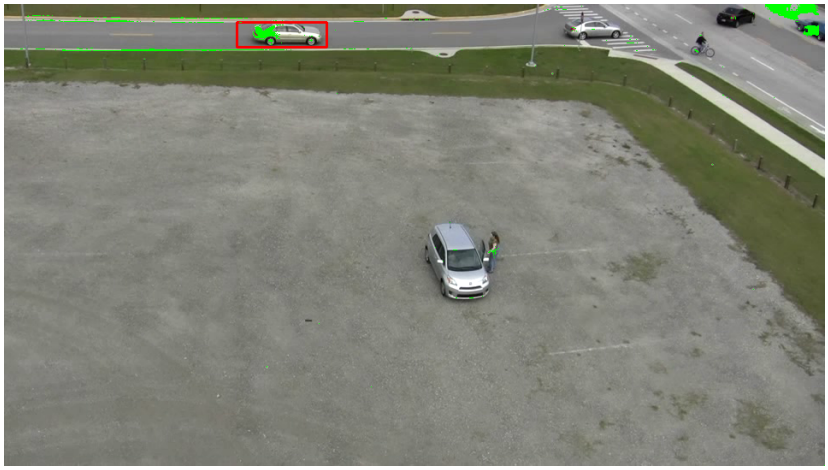
Obrázek 3.1: Experiment č. 1, snímek č. 1.

Obrázek 3.2: Experiment č. 1, snímek č. 2.

Obrázek 3.3: Experiment č. 1, snímek č. 3.

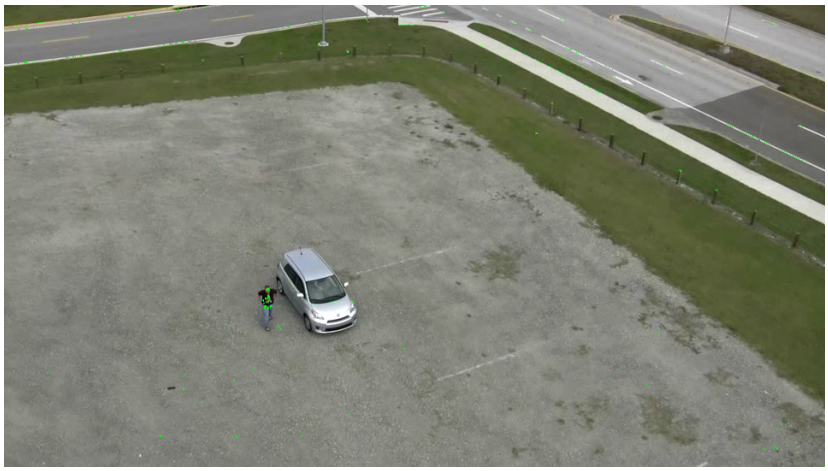


Obrázek 3.4: Experiment č. 1, snímek č. 4.



Obrázek 3.5: Experiment č. 1, snímek č. 5.

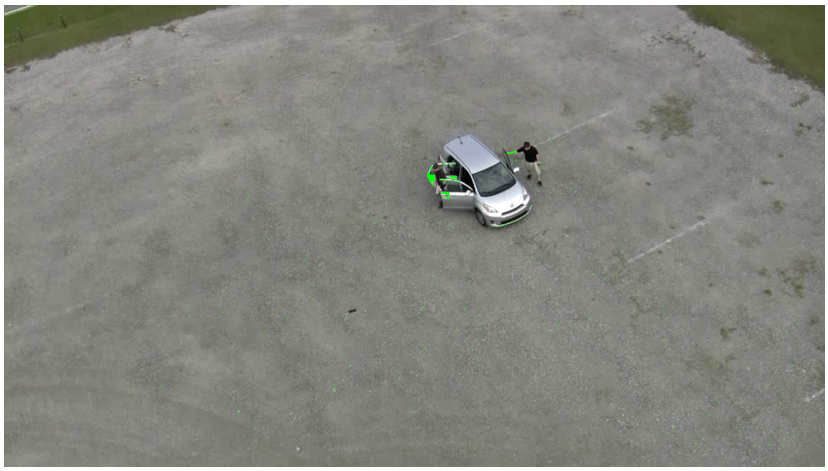
Obrázek 3.6: Experiment č. 1, snímek č. 6.



Obrázek 3.7: Experiment č. 1, snímek č. 7.

Obrázek 3.8: Experiment č. 1, snímek č. 8.

Obrázek 3.9: Experiment č. 1, snímek č. 9.



Obrázek 3.10: Experiment č. 1, snímek č. 10.

Obrázek 3.11: Experiment č. 1, snímek č. 11.

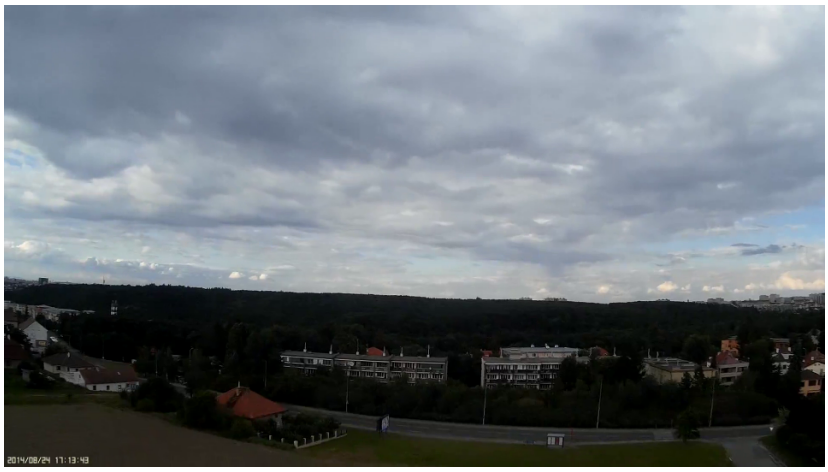
Detektor	Pohyb detek. (TP)	Pohyb nedetek. (FN)	Chybně detek. (FP)
Příznakový	8	22	4
Zbytkový	26	4	7

Tabulka 3.1: Výsledky experimentu č. 1.

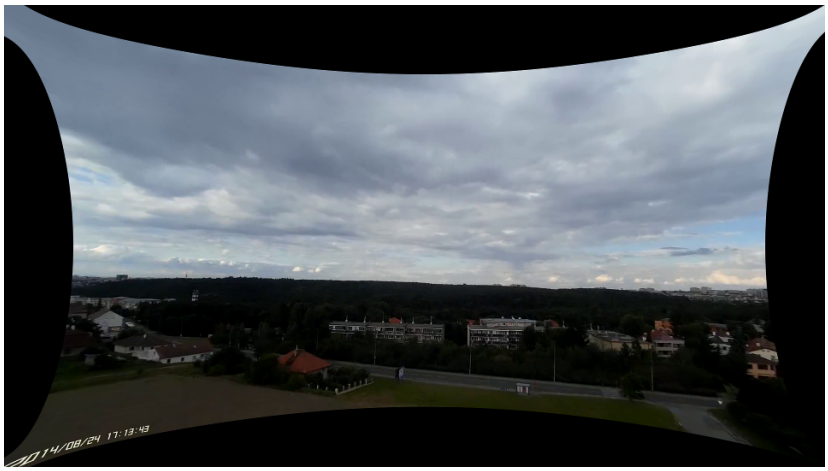
Pokud jsou porovnány oba detektory v tomto experimentu, tak Příznakový detektor detekoval mnohem méně pohybujících se objektů než Zbytkový detektor, protože některé pohybující se objekty byly příliš malé a nebyly na nich detekovány žádné příznaky. V tabulce 3.1 je odhad výsledku, který byl počítán manuálně z obrázků.

3.2 Experiment č. 2

U tohoto experimentu bude analyzováno video, které bylo poskytnuto panem Chudobou. Video bylo natočeno nekvalitní kamerou, proto je nutná kalibrace kamery (obrázky: 3.12 a 3.13). Dochází k velkým vibracím, proto se na videu znatelný velký šum. Na videu je možné pozorovat pohyblivé objekty: projíždějící auta a autobus (obrázky: 3.14 až 3.18). Délka videa je 1 minuta 41 sekund, frekvence snímků je 30 Hz a rozlišení je 1920 $\times$ 1080, které bude sníženo na 960 $\times$ 540. Pro detekci pohybujících se objektů bude spuštěna pouze metoda zbytková analýza, protože pohybující se objekty jsou příliš malé a proto je příznakový detektor není schopný detekovat.
Pro zajištění korespondence snímků byla využita příznaková metoda pomocí SURF příznaků, kde byl zvolem $hesianTreshold = 100$ a pro funkci warp bylo vybráno 30% spárovaných příznaků s nejmenší odchylkou.
Pro Zbytkový detektor byl zvolen $distanceTreshold = 7$.



Obrázek 3.12: Experiment č. 2, nezkalibrovaný snímek.



Obrázek 3.13: Experiment č. 2, zkalibrovaný snímek.

Obrázek 3.14: Experiment č. 2, snímek č. 1.

Obrázek 3.15: Experiment č. 2, snímek č. 2.

Obrázek 3.16: Experiment č. 2, snímek č. 3.

Obrázek 3.17: Experiment č. 2, snímek č. 4.

Obrázek 3.18: Experiment č. 2, snímek č. 5.

V tomto videu nebylo příliš pohybujících se objektů, ale zbytkový detektor téměř vždy všechny odhalil. Bohužel pokud nebyly snímky dobře zkorespondované, bylo detekováno velké množství zbytku v místech, kde se nacházejí budovy a nebe.

Kapitola 4

Závěr

Byl analyzován problém detekce pohybujících se objektů ve videu, na základě čehož bylo navrženo řešení tohoto problému. Jako programovací jazyk byl zvolen jazyk C++ a byly využity volně dostupné knihovny OpenCV a OpenCL. Naimplementovaná aplikace byla rozdělena do několika bloků, které mezi sebou komunikují a přeposílají si data. Tento způsob implementace je výhodný v tom, že je přehledný a jednotlivé bloky se dají jednoduše vylepšovat aniž by se musel přepsat primární algoritmus.
Naimplementovaný algoritmus se skládá ze tří částí: kalibrace kamery, korespondence snímků a detektor změn.
Původním cílem byla implementace real-timového systému detekce pohybujících se objektů přímo na hardwaru drony, bohužel problém detekce pohybujících se objektů je velice komplikovaný a nebyl dostatek dostatek času k implementaci pokročilejších algoritmů, které využívají model 3D prostředí vytvořený z jednotlivých snímků. Vytvořená aplikace je funkční (koresponduje snímky a detekuje pohybující se objekty), ale nefunguje jako real-timový systém a není schopna detekovat všechny převážně malé pohybující se objekty. Proto bude potřeba této práci věnovat ještě hodně úsilí a času a danou aplikaci vylepšit a optimalizovat, protože korespondence snímků je příliš pomalá a bylo by výhodné tento problém přenést na GPU. To samé platí pro detektory, který pracují pomalu a nedetekují všechny pohybující se objekty.
Byly provedeny dva experimenty, ve kterých byla dokázána funkčnost implementovaného algoritmu. Příznakový detektor detekuje belké pohybující se objekty s pravděpodobností $P \approx 27\%$ a chybou $F \approx 30\%$, Zbytkový detektor detekuje pohybující se objekty s pravděpodobností $P \approx 85\%$ a chybou $F \approx 23\%$. Aplikace funguje mnohem lépe, pokud je video kvalitní a nedochází k velkým pohybům kamery. Zbytkový detektor funguje lépe než Příznakový detektor, protože SURF příznaky se většinou nedetekují na objektech, které jsou hledány. Pro vylepšení detekce pohybujících se objektů je třeba nejdříve vylepšit a zpřesnit korespondenci snímků, protože i malá odchylka dokáže překazit detekci.

Literatura

[1] arihant lunawat. Camera calibration and fundamental matrix. http://www.cc.gatech.edu/~hays/compvision/results/proj3/alunawat3/index.html.

[2] G. Bradski and A. Kaehler. Learning opencv. https://learningopencv.wordpress.com/2010/05/23/opencv-structure-and-content-overview/.

[3] OpenCV comunity group. http://opencv.org/.

[4] OpenCV comunity group. Camera calibration with opencv. http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html.

[5] OpenCV comunity groupp. Camera calibration with opencv. http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html.

[6] GITTA. Quantitative choropleth maps. http://www.gitta.info/ThematicCart/en/html/TypogrDesign_learningObject4.html.

[7] Google. Digitalizace rastrového obrazu. https://sites.google.com/site/xgrafika/digitalizace-rastroveho-obrazu.

[8] Khronos OpenCL Working Group. The opencl specification. https://www.khronos.org/registry/cl/specs/opencl-1.1.pdf.

[9] Václav Hlaváč. Počítačové vidění vs. digitální zpracování obrazu: Digitálníobraz a jeho vlastnosti [online]. University Lecture, 2012.

[10] Itseez. *The OpenCV Reference Manual*, 2.4.9.0 edition, April 2014.

[11] Rüdiger Dillmann Pedram Azad, Tilo Gockel. Computer vision - principles and practice, 2008.

[12] Ing. Roman Pihan. sférická vada. http://fotoroman.cz/glossary2/2_sfera.htm. Accessed: 2011.

[13] Khalid Sayood. Introduction to data compression, third edition, 2005.

[14] Doc. Miloš Steinhart. Optika - geometrická optika. http://webak.upce.cz/~stein/lectcz/ffsz_08.html. Accessed: 2011-01-4.

[15] Wikipedia. Color — Wikipedia, the free encyclopedia, 2016. [Online; accessed 30-March-2016].

[16] Wikipedia. Lens (optics) — Wikipedia, the free encyclopedia, 2016. [Online; accessed 5-May-2016].

[17] Wikipedia. Median filter — Wikipedia, the free encyclopedia, 2016. [Online; accessed 23-May-2016].

[18] Wikipedia. Noise reduction — Wikipedia, the free encyclopedia, 2016. [Online; accessed 23-May-2016].

[19] Wikipedia. Odstranění šumur — Wikipedia, the free encyclopedia, 2016. [Online; accessed 18-May-2014].

Obsah CD

Přiložené CD obsahuje zdrojové kódy pro ... text diplomové práce ve formátu PDF a zdrojové kódy celého textu pro systém LaTeX. V následující tabulce je popsána struktura CD.

Adresář	Popis
`src`	zdrojové kódy knihovny
`doc`	zdrojové kódy textu diplomové práce
`thesis.pdf`	text diplomové práce

Tabulka 1: Adresářová struktura na CD